Interactive Inspection of Segmented Volume Models

Doctoral Thesis

Jordi Surinyac i Albareda

Advisor: Pere Brunet i Crosa

Universitat Politècnica de Catalunya
BarcelonaTECH

Departament de Ciències de la Computació

Ph.D. Program in Software

Barcelona December 2015

Abstract

Present systems for the acquisition of medical images are providing better and better resolution images which end up in bigger amounts of data, even in scenarios with learning or training purposes. Interactive inspection of segmented volume models and anatomy atlases are becoming more popular as 3D medical models are improving in quality and detail.

The use of low-end, portable devices is gradually spreading due to their portability and easy maintenance. There is a user interest on displaying the atlases and interacting with them in such devices, but they do not have the storage requirements to manage big amounts of data. For this reason, client-server architectures are used. However, transmission time for the volumetric information and low performance hardware properties make quite complex to build efficient visualization systems on these devices. Therefore, advanced new algorithms for compression and progressive transmission are required.

In this work is presented an entire framework that defines the following elements: an efficient data structure for segmented volume models based on a hierarchical data structure of surfels per anatomical structure, and a novel technique for the progressive transmission and region-based inspection of medical models.

The volume models are assumed to be already segmented, process which is beyond the scope of this work. Every segmented organ in the region of interest is preprocessed and represented by an octree of surfels with code normals and color information. Our scheme is based on a forest of multiresolution octrees. Anatomy atlases are represented as forests of surfel octrees, supporting local interaction in the client device and selection of groups of anatomical regions.

Client-Server model transmission requires good compression techniques, being as loss-less as possible and having a high compression rate, to obtain a reduced data flow through the network. Surfel Octrees are compact enough for transmission through networks with limited bandwidth, and provide good visual quality in the client devices at a limited footprint. Individual octrees can be progressively transmitted to the client on demand, depending on the importance of the organs and on specific interaction queries.

The proposed approach is discussed by presenting several examples in low-end devices such as mobiles and tablets. We analyze the performance of the presented scheme, the memory requirements and the usability of several interactive inspection tools.

Resum

Els sistemes actuals d'adquisició de dades mèdiques proporcionen imatges de cada vegada més resolució, havent-se de gestionar grans quantitats de dades fins i tot en àmbits d'ensenyament o formació. La inspecció interactiva de models de volum segmentats i d'atles anatòmics són cada vegada més populars a mesura que els models de volums mèdics en 3D estan millorant en qualitat i detall.

La utilització de dispositius portàtils de gamma baixa s'està estenent gradualment per la seva facilitat d'ús i de manteniment. Hi ha interès per part dels usuaris en la visualització i interacció amb atles mèdics en aquest tipus de màquines. Però aquests dispositius no estan capacitats per emmagatzemar grans quantitats de dades i per tant es fan servir arquitectures client-servidor. Per aquest motiu, tenint present el temps de transmissió de la informació volumètrica i les escasses capacitats del hardware, fa que sigui complicat construir sistemes de visualització eficients en aquests dispositius. Calen nous algorismes avançats de compressió i de transmissió progressiva de dades.

En aquesta tesi es presenta un entorn complet que defineix: un model de dades eficient per a models segmentats de volum, basat en una estructura de dades jeràrquica de surfels per cada estructura anatòmica, i una tècnica innovadora per a la transmissió progressiva de models mèdics i d'inspecció per regions d'interès.

Se suposa que els models de volum ja estan segmentats, procés que està més enllà de l'abast d'aquest treball. Cada òrgan segmentat en la regió d'interès és preprocessat i es representa amb un Octree de Surfels que conté les normals codificades i informació de color. El nostre sistema es basa en un bosc d'octrees multiresolució. Un atles anatòmic es representa com a un bosc d'Octrees de Surfels, que permet la interacció i selecció de grups de regions anatòmiques en el dispositiu client de manera local.

El model de transmissió Client-Servidor requereix bones tècniques de compressió de dades, que tinguin un índex de pèrdua d'informació tan baix com sigui possible i alhora una taxa de compressió suficientment alta com per que hi hagi un flux prou reduït de dades en la xarxa. Els Octrees de Surfels són

suficientment compactes com per ser transmesos a través de xarxes usant una amplada de banda reduïda però proporcionant una bona qualitat visual en els dispositius client. Octrees individuals poden ser transmesos progressivament al client sota demanda, en funció de la importància dels òrgans i de les consultes d'interacció especfiques.

L'enfocament proposat s'analitza mitjançant la presentació de diversos exemples en dispositius de gamma baixa com mòbils i tauletes. Estudiem el rendiment de l'esquema presentat, els requisits de memòria i la usabilitat de diverses eines d'inspecció interactives.

Acknowledgements

The writing of this thesis is not the work of only one person. Some people, in different ways, have contributed from the very beginning to the conclusion of the process. This section is dedicated to the recognition of all those involved in this work.

The guidance and assistance given by my advisor Dr. Pere Brunet have been fundamental in the development of the thesis. He envisioned the direction to follow at every crossroads and he knew how to put the pieces together. This is my thesis, but it is also a product of his work. Beyond the technical perspective, he always showed patience with me when ideas were not easy to understand and furnished me with encouragement and advice in the moments of no progress. It has been a pleasure to learn from him. He deserves my sincere gratitude.

I thank to the people from the Departament de Ciències de la Computació, particularly to Isabel Navazo, Eva Monclús, Pere-Pau Vázquez and Toni Chica, for their kindness and the generous help they provided when I asked for.

I must also show gratitude to my family -relatives and in-laws- for their support throughout all the time. Especially to my parents for their determination to provide higher education to their sons, despite all difficulties.

Finally, I dedicate this thesis to my wife, Laura and my daughter, Maria. They helped me in the best way they can do: with understanding, unconditional love and time. It is the moment to give back to them some of what they have given to me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Medicine related professionals and students must have a thorough knowledge of the human body anatomy. Therefore, they need to learn all the details of the anatomical structures (*organs* in the following) and their inner constitution. The best didactic tool is, in any sense, a real body. Corpses allow any type of dissections and manipulations to study their interior. However, the availability of corpses is very limited: they cannot be obtained at any moment nor at any place.

To solve this issue, anatomic Atlases have been the usual alternative. They have adopted different forms through the evolution of medicine and technology. From their beginning and for many years they were in the form of handmade drawings. The posterior appearance of photography increased the image realism, even though handmade drawing still remains since it allows the highlighting of features.

Digital Atlases have appeared with the spread of computers. They have the advantage of 3D navigation. The user can inspect the body from different points of view, which became a novelty in anatomical Atlases because it better simulates the real inspection of the body. Digital Atlases also support links, metadata and other sources as X-ray images and sonograms, which assist and improve the learning process of the student.

Newer Atlases allow interaction with organs, reproduce articulation movement and deformations. If the physical behavior of organs is included in the model, simulations of surgery can also be done.

The geometry's accuracy of digital models is variable, ranging from artists' drawings to exact samples from CT or other acquisition devices. Using the latter approach, color does not need to come from the artist's palette. Like photographs, it can be real.

Digitized bodies are massive models, too large to be processed in real-time. Auxiliary preprocessed data structures are necessary for interactive rendering.

Software schemes for storage and visualization of human models, which preserve real colors and positions of anatomic elements, are required to make an Atlas an effective tool.

Portable computers and low-cost devices add another challenge to digital Atlases: their lower memory and CPU/GPU capacity force programmers to design new schemes of visualization and interaction. Nowadays they still remain a topic of interest.

In the following sections the work done in the present thesis is described. Section 1.1 states our motivations and the goals of the research project. Section 1.2 summarizes the main contributions and Section 1.3 details the structure of this document.

1.1 Motivation And Objectives

Following the previous remarks, we want to design a new system able to construct an anatomical Atlas from acquired data that meets the following properties and features:

- **Accuracy.** Medical educational models must be as close as possible to a real body. Geometry and color of anatomical structures have to be realistic and faithful to the original model.

- **Low memory footprint.** Technical improvements make memory storage always grow over time but medical scans, being mainly volumetric, consume a big quantity of data and it is easy to exceed any devices' limits. Low-cost portable devices do not usually have big amounts of such resources. Therefore, the model must be structured and compressed to fit in small memories. Moreover, if the model is transmitted through a network, less data means less bandwidth occupation and quicker loading times at the client side.

- **Multiresolution.** The utilization of a level of detail (LOD) scheme is desirable in our work because it allows emphasizing regions of the model, rendering distant regions without unnecessary geometry, and also loading the model at a low resolution, which leads to the utilization of larger models.

- **Interactivity.** Even though real-time is not sought, sluggish visualizations and manipulations of the model decrease the usability of the tool. Algorithms and techniques must be efficient enough to permit interactivity with the user.

1.2 Contributions

The research of this thesis is aimed at three main objectives:

- The study of local surface representations without topology, as a tool for the efficient encoding of the boundary of segmented medical structures coming from discrete binary volume representations.
- The analysis of hierarchical data structures to represent the surface of anatomical structures.
- The design of an interactive inspection scheme in client-server systems, with the objective of visualizing organ boundaries and detailed internal structures.

In the rest of this Section, we present a more detailed description of the objectives.

Hypothesis statement

Our main research hypothesis is that a discrete and hierarchical surfel representation is efficient and suitable for the interaction with anatomic atlases in distributed applications with low-end client devices.

Local surface representations

Medical Atlases are usually generated from segmented data. The segmentation of the volume creates a discrimination of anatomical structures by explicitly representing their boundaries. As medical experts are usually and primarily interested in the visualization of organ boundaries, we analyze the representation of 2D manifold surfaces, and we specifically consider the case of local surface representations.

To display boundaries of medical structures, we consider that visualization schemes with independent local geometric primitives are preferable, since no topology is required to be included in the model. It also allows the execution of processes that work locally. Given that Atlases come from volumetric models, our contribution consists in taking advantage of spatial subdivisions of the volume to build very compact local representations of surfaces of organs.

Surfels are a widely studied topic (see Section 2.5) but the variation presented in this work uses a hierarchical spatial subdivision to complement their geometry, needing less information to be determined. A faithful method based in tetrahedra and linear fields to encode the supporting plane is used. A realistic color scheme for surfels is also presented. Chapter 3 is devoted to the

description of constrained surfels and how they are encoded. The obtained results were published in:

A Client-Server Architecture for the Interactive Inspection of Segmented Volume Models
Jordi Surinyac and Pere Brunet.
XXIII Congreso Español de Informática Gráfica (CEIG), 2013, Madrid, Spain, pages 99-108

Even though the main interest of users is surface visualization, the representation of the inner structures of organs is also their concern. A proposal is made in that direction, where an arbitrary plane cuts the organs in the scene. Chapter 5 explains the details of sectioning among other concepts.

Surfel Octrees

The Atlas representation must support global and local visualizations of organs as well as other interactive manipulations. Moreover, it is well known that the requirements listed in Section 1.1 can be satisfied by hierarchical data structures like octrees. We therefore propose Surfel Octrees as the data structure for Constrained Surfels.

Surfels contained in lower level nodes maintain fidelity to the geometry of the original samples, but they also add smoothness to improve the realism of anatomical structures. These surfels are simplified to obtain the surfels of upper level nodes, adding the constraint of volume preservation. Every octree level is a LOD of the model. Constrained Surfels and Surfel Octrees are represented in a very compact method to reduce storage requirements and to speed up network transmission. In Chapter 4 the details of octree creation and compression are described. These results were published in:

Surfel Octrees: A New Scheme for Interactive Inspection of Anatomy Atlas in Client-Server Applications
Jordi Surinyac and Pere Brunet.
XXV Congreso Español de Informática Gráfica (CEIG) 2015, Benicàssim, Spain, pages 61-70

Interactive inspection in client-server systems

We have observed that digital Atlas requirements from medical experts include interaction tools like rotating, translating, zooming and selecting at interactive rates. While this is easy to accomplish in powerful computers that contain the whole model in their memory and transfer speed is not an issue, it becomes

a key concept in client-server architectures where user interaction and model storage take place in different computers.

Our interactive proposal takes into account these four concepts:

- An interactive tool for examining the inner structure of organs. The client can receive sections of the model on demand and the user can inspect them off-line.
- Only required data are transmitted in order to minimize bandwidth occupancy. Users can select or de-select anatomical structures to be displayed, so transference is client-driven.
- It features a multiresolution scheme for the anatomical structures. Again, the user has control about the level of detail of the model to be transmitted and rendered.
- It uses a texture scheme for rendering that minimizes the amount of data transmitted. The color of the model is obtained from textures which are sent at the beginning of the session. During interaction no color information is dispatched from the server.

A user evaluation test is being performed to obtain external feedback about this framework. The results of this work will be submitted for publication as a research paper.

1.3 Thesis Outline

The present document is structured in six chapters apart from the current opening one:

- **Chapter 2: Client-Server Inspection of Volume Models**, reviews existing literature and the state of the art regarding the several topics considered in this work: anatomical Atlas, volume models, hierarchical volume representation, compression, surfel representations, client-server architectures and low-cost portable client devices.
- **Chapter 3: Surfels and Space Subdivisions**, analyzes the surfels as a local surface representation and presents constrained surfels as visualization primitives. A hierarchical space subdivision contributes to their localization, so less geometrical information is needed to be stored with them. It also presents the Connected Cube Plane parameterization which provides a useful way to encode the plane that supports the surfel. Finally, four methods to generate the color of the surfels are analyzed.

- **Chapter 4: Surfel Octrees**, describes the hierarchical data structure that contains the surfels. It also details the generation of high-resolution surfels and their later simplifications, which populate the different levels of the octree. Finally, it details an efficient compact representation of both surfels and the octree.

- **Chapter 5: Results and Discussion**, covers the following concepts: the client-server architecture approach, including the preprocess step that creates the forest octree of anatomical structures and the transmission scheme; the interaction paradigms we have studied and implemented; the visualization of a volume with sections; the interactive multiresolution model; results and their discussion; and a final evaluation.

- **Chapter 6: Conclusions and Future Work** closes this dissertation with a general review of the work done during this thesis and highlights the achieved goals. Some lines of future work are presented to extend the research done.

Chapter 2

Client-Server Inspection of Volume Models

This work is aimed to the surface extraction from a medical volume model and its posterior visualization in low-end devices. Surface extraction and visualization of large volumes of data have been relevant topics of interest for a long time and have generated a large bibliography devoted to them. Moreover, the expansion of portable devices along with the recent improvements of their storage and graphical capacities, enable them to be the front-end elements for visualization and user interaction. A growing wave of bibliography begins to take into consideration this new actor in computer graphics. This Chapter reviews significant works related to the problems addressed in this thesis.

2.1 Anatomical Atlas

Anatomical Atlases have been linked to medical practice from its very beginning. Early physicians, teachers to next generations of physicians used illustrations to convey their knowledge, so texts and drawings became their legacy. Often physicians made the drawings by themselves, but sometimes they needed the help of talented artists. Without no intention of being exhaustive, we present a short review of the main ancestors to modern Atlases outlying the innovations they introduced.

There are a number of specific Atlases: embryology, pediatrics, muscular, skeletal, dentistry, locomotive system, physiology... This review is only focused in general anatomical Atlases.

It is surprising that early physicians, such as Hippocrates (460-370 B.C.), Herophilus (330-260 B.C.), Erasistratus (304-250 B.C.) and even Galen (130-200 A.D.) did not illustrate their observations with any drawings or sketches.

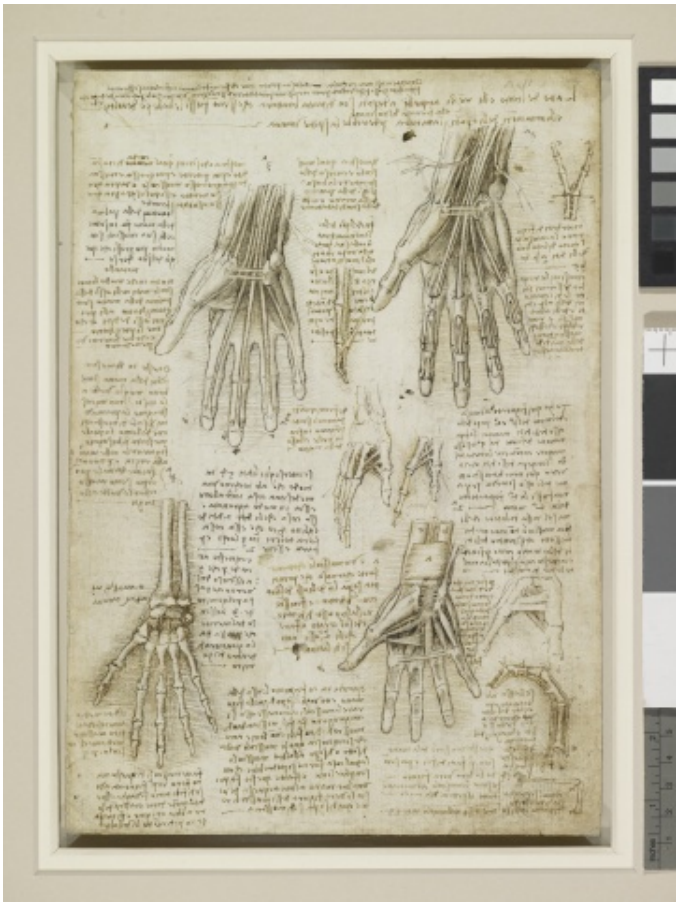


Figure 2.1: *Recto: The bones, muscles and tendons of the hand. Verso: The bones of the hand*, Leonardo da Vinci. RCIN 919009, Royal Collection Trust / © Her Majesty Queen Elizabeth II 2015

And those who did it probably had their work erased in Alexandria's Library fire. Moreover, the taboo on human dissection and religious restrictions -until 1537- left us without early paintings of the inside of the human body.

The notes and drawings about anatomy of Leonardo da Vinci (1452-1519) were surprisingly precise. He prioritized realism and used cross-sections and views from multiple angles. Figure 2.1 shows muscles and tendons of the hand with unprecedented accuracy.

Bartolomeo Eustachi (circa 1500-1574) was assisted by artist Pier Matteo Pini in the making of his long-used -two centuries- anatomical illustrations. To achieve more precision in the commentaries of the drawings, rulers were placed around the images in a rectangular pattern allowing to locate the details with

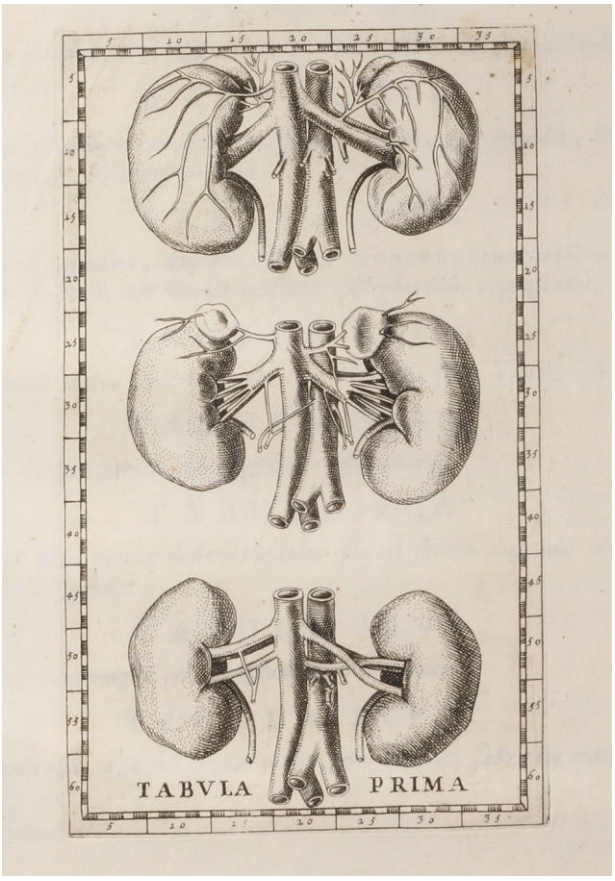


Figure 2.2: Kidneys, 47th plate, *Tabulae Anatomicae*, Bartolomaeus Eustachius. Courtesy of Linda Hall Library of Science, Engineering & Technology. Note the rulers around the figures.

coordinates, as we can see in Figure 2.2.

Andreas Vesal (1514-1564) corrected many of Galen's anatomical assumptions. He published *De Humanis Corporis Fabrica* for his students, which was reprinted many times. The drawings were performed by himself and the artist Pietro da Cortona. Bodies often appear in allegorical poses. Figure 2.3.

Bernhard Albinus (1697-1770) was not the author of his drawings -it was Jan Wandelaar-. However, he thought of placing a net, a grid pattern, between the model and the artist's point of view to increase the accuracy of the drawings.

Hieronyumus Fabricius (1533-1619) emphasized the fact that the plates in

Figure 2.3: *De humani corporis fabrica librorum epitome*, Folio 9v, Andreas Vesal. Courtesy of University of Glasgow Library, Special Collections.

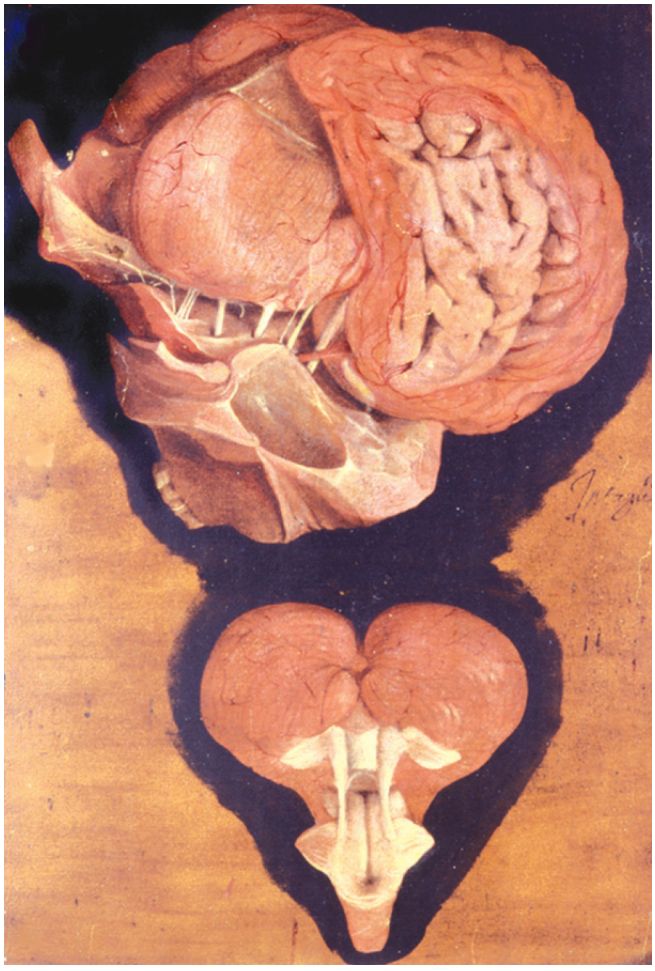


Figure 2.4: *Tabulae Pictae*, plate 112.10, Hieronymus Fabricius.

his work *Tabulae Pictae* [7] were represented in their real size and colors, as we can see in Figure 2.4.

Max Brödel (1870-1941) was not a physician but a medical illustrator. He used lithography as a means to make tissue seem alive since other techniques -crayon, pencil, ink, watercolor, pastel- produced a plastic-like effect. He also created a sense of depth by using an eraser to lift highlights and soften the edges.

Anatomical Atlases appear in the twentieth century . They were published as books as we currently know them. Their titles only have a few variations and they are broadly known by the name of the main author.

Anatomy Descriptive and Surgical [8], published in 1859, renamed *Gray's*

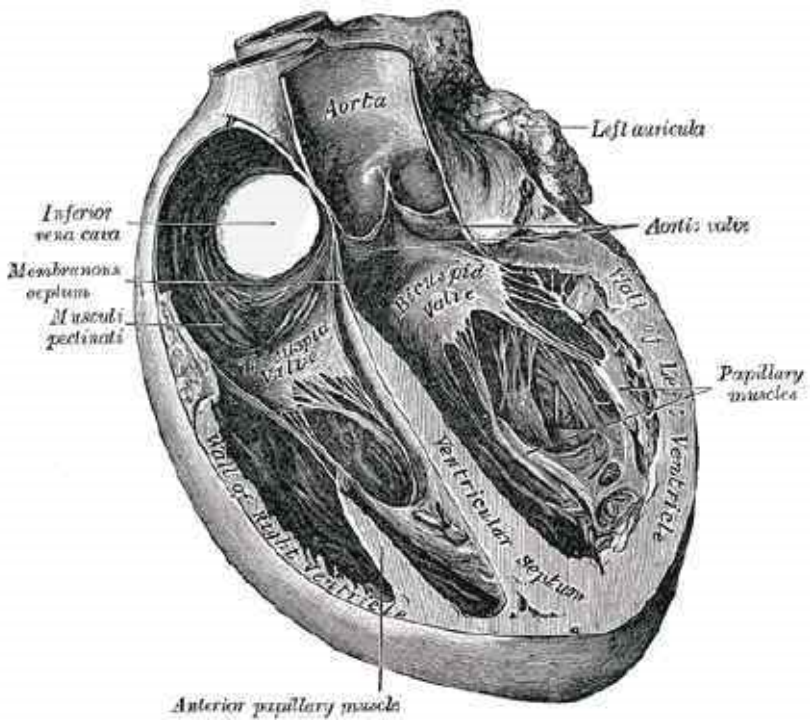


Figure 2.5: Heart section from the 1918 edition of *Gray's Anatomy*.

Anatomy in 1938, is even today a very influential textbook of anatomy from Gray and illustrations from Carter. In 2008 it reached the 40th edition. Figure 2.5 portraits an original drawing.

Netter [17] is a physician that had also studied art before he started medicine. His sketches style in the classroom attracted the attention of the medical faculty and began to illustrate articles and textbooks. In 2005, Elsevier purchased all his collection of drawings and edited more than 50 books with his illustrations.

The *General Anatomy and Musculoskeletal System, Thieme Atlas of Anatomy* from Shuenke et al. [24] shows realistic drawings along with schemes.

Portraying non-realistic drawings -the coloring does not intend to be real- is not a limitation on the use of Atlases. *Sobotta Atlas* [20] (Figure 2.6) has reached the 15th edition, *Grant's Atlas of Anatomy* [2] the 13th edition and *A Regional Atlas of the Human Body* from Clemente [6], the 6th.

Nowadays many of these Atlases have digitized versions.

Only recent Atlases are based on photographs. They can also be found in digitized versions and can be used in portable devices.

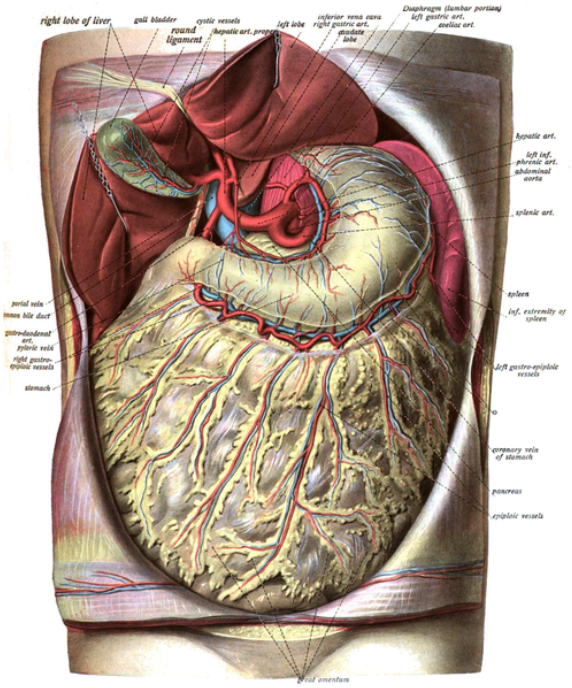


Figure 2.6: Non-realistic drawing from *Sobotta Atlas*, from the 1918 edition.

Rohen et al. [21] created an Atlas with photographs and other media: Computed Tomography (CT) scans, Magnetic Resonance Imaging (MRI) and handmade schemes. In some photographs, organs are hand-painted to highlight them. Organs can be shown isolated from the environment, embedded in it, or sectioned. Moses et al. [15] also portraits painted organs.

Videos have also been used, but their use in Atlases is not widespread since interaction with the user is worse than books or computers. Videos have been reserved to medical procedures. Many of the videos are animations (by hand or generated by Computer Graphics).

The Nebraska Medical Center (which includes a medical university) among others created several animations specialized in their areas of interest [18].

Acland's Video Atlas of Human Anatomy [1] consists of video footage. The camera moves around the object of study and gives the user a 3D sensation.

Blausen [5] produces CG animations of physiological functions but also features a simple 3D anatomical Atlas. It is suitable for tablets, smartphones, web pages and also as video-on-demand.

Digital 3D Atlases allow annotation, coloring of structures, hyperlinks and more. Initially they were designed as standalone computer applications, but nowadays they are moving to web applications -which can run in browsers or

client applications- and portable devices.

The Visible Body [26] and Kenhub [13] feature artistic representations of the anatomy. They also include animations of physiological functions and procedures. Visible Body is designed as an application but Kenhub is a web application. The *Zygote Body* [27] was formerly *Google Body* before Google decided to move its maintenance to Zygote Media Group. It is a web application and supports dynamic sectioning of selected structures. The BioDigital Human [4] is designed as a web application and runs in a browser or as an application on portable devices. It provides its own API to integrate the core navigation to other host programs.

To visualize interior and exterior parts of a volume at the same time Monclús et al. [16] propose a metaphor named Virtual Magic Lantern which uses a 3D pointing device to define a visualization cone. The volume outside the cone is rendered using the original Transfer Function whereas the inner one can use a different function, for instance rendering only specific organs or parts of the model. Although this metaphor was designed for a raycasting algorithm, it can be adapted to other visualization methods.

The Visible Human Data Set [22] is a source of a number of applications that transform raw data into anatomical Atlases, such as Anatomia Creations [3] or ToLTech [25]. The most relevant is the VOXEL-MAN project, with some regional Atlases [10, 9] along with surgery and dental simulators. Lin et al. [14] visualize selected parts of the model using environments. A survey of applications using the dataset can be found in [12].

The Interactive Institute Swedish ICT developed the Inside Explorer [11], a visualization software that uses direct volume rendering and operates with CT or MRI segmented medical data. Together with the Virtual Autopsy Table, a collaborative touchscreen table, many museums use this system in their exhibition rooms.

The *Tuvok* architecture from Fogal and Krüger [175] was designed to visualize very large datasets, in old and new GPUs, with cross platform support. To achieve interactive visualization, a progressive LOD and a lowering of the screen resolution of the rendering is used. Also, a specific paging strategy is devised to cache efficiently the bricks -chunks of data that fit in the texture memory of the GPU-.

Medical data models can be used to build real objects with 3D printers. Sakata et al. [23] use data from a specific patient to create an elastic liver to reproduce in vivo conditions and enable preoperative simulations. Perez et al. [19] use CT scan data to create a real 3D model of the fractured acetabular part of the pelvis of a patient to provide surgical simulation and to preconture the plates they will implant during the surgery.

2.2 Volume Models

Medical scanners are designed to penetrate into the body. These devices perform a depth scan of the body, resulting in a parallelepiped volume model. A volume model consists of an ordered set of samples with their geometric position in space and one or more measured data properties. The most usual arrangement of the ordered set is a 3D grid, which can be seen as a sequence of 2D matrices of samples, also known as slices. The region of space associated with a single sample is called voxel.

Being of volumetric nature, these models commonly contain a big number of voxels. Working with them can be very expensive in terms of memory and CPU/GPU time. Systems that process these models utilize specifically designed data structures and algorithms to deal with great amounts of data.

A volume model is a discrete representation of a scalar field $V(x, y, z)$ in a space. The values are usually sampled in a regular [31, 50, 49, 54, 43] or an irregular grid [28, 40, 60]. The former produces a voxelization or a voxel model.

2.2.1 Interactive Visualization

The visualization of a volume model consists in the 3D visualization of the shapes contained in the model in a way such that their values (or ranges of them) are shown while displaying their geometry. Naive projection of the color of all voxels into the framebuffer would result in a mixture of colors where no information can be retrieved by the user, so the method of visualization has to be carefully devised. Two schemes are used, depending on the objective desired:

- *Direct Volume Rendering* (DVR). Values of voxels -color and opacity- are projected onto the screen, blending them when two or more coincide in the same pixel or area. Values of voxels are assigned to colors via a transfer function. Several rendering techniques use this scheme: 3D textures, from Hibbard and Santek[30]; Levoy [32] introduced volume ray casting -although he described it as ray tracing-; splatting -as in the sound snowballs make when hitting the wall, as Westover [34] himself graphically described-; and Lacroute and Levoy [33] applied a shear to the slices (and also a scale in the case of perspective projection), projected the viewing rays to a plane and finally warped them according to the image plane, producing the shear-warp method.

- *Indirect Volume Rendering* (IVR), also known as *Surface Fitting*. Only voxels with a given range of values are rendered. A segmentation pre-

process generating a point set [28] or an isosurface (like Marching Cubes algorithm [31] or Marching Tetrahedra [29]) is usually performed in a first step, so that the final rendering visualizes this intermediate geometric structure (surface or points).

In this work we take the latter approach, so a more detailed review of IVR methods is made.

2.2.2 Segmentation

Binary volumetric models have become more important over the last few years. Binary voxelizations are commonly generated as a result of segmentation algorithms working with volumetric medical data (see for instance [38]) that assign each voxel to a specific organ. Binary volume models miss information on the scalar field of densities and they only represent in/out binary information on each of the segmented organs. Discrete volume objects are also created by the voxelization of different input models. Many reconstruction [36] and model repair [35] algorithms are based on volume binary models, and a number of volume operations like splitting produce binary information in the modified regions. Anatomy Atlases are based on binary models since the main interest of the user is the identification and differentiation of anatomical structures. It must be clear to which organ a given voxel belongs to.

The Visible Human dataset [22] was the data source of Kang et al. [37] for their semi-automatic method of segmentation, which consisted of a first step where radiologists drew contours in slices at specified intervals, and then an interpolation process that automatically found the contour in intermediate slices.

Brain Atlases are used to register the brain's images of a specific patient with the Atlas, so that a clinical comparison can be made. Seixas and Silveira [42] contrasted the three main methods of segmentation used in brain Atlases. Soldea et al. [41] analyzed the efficacy of using pairs of Atlases at the same time.

Zhang et al. [39] used a previous work [40] as a basis to obtain a tetrahedrization of the model, constructing a global mesh that separates each area and defining correct boundary isosurfaces. In a boundary cell of two areas, the points of intersection of the isosurface and their normals are used, with a quadratic error function, to obtain a vertex (and its corresponding edges and faces). With more areas, more vertices can be computed. Another quadratic error function finds the optimal minimizing point, which is used as a common vertex in all areas. The rest of the algorithm followed the rules of [40].

2.2.3 Surface Extraction

Surface extraction has been a relevant topic of interest for a long time and has generated a large bibliography devoted to it. This Section reviews significant works related to the problems addressed in this thesis.

The papers reviewed in this Section can be classified in different ways according to a number of independent criteria. Data represented in the volume model can be binary, values from a scalar field or Hermite data. The structure of the volume cells can be regular (voxels) or irregular (tetrahedron). Isosurface extraction can be based on direct (Marching Cubes) or dual algorithms, and ambiguities are solved in different ways. Some of the presented approaches are well suited to non-uniform, hierarchical space partitions.

Given the variety of shapes that we will encounter in medical objects, most of the presented ideas and algorithms will be taken into account. The main focus will be, however, in isosurface extraction with feature preservation of non-regular hierarchical space subdivisions.

Marching cubes

The Marching Cubes algorithm (MC) [31] from Lorenson and Cline extracts a triangular surface from a volume model. The configuration of vertices of a voxel (in/out) is the key to access a look-up table and, if the voxel stabs the boundary of the model, a triangulated mesh is constructed in that voxel. This is a simple method of surface extraction but it has several drawbacks, one of them being the ambiguous cases that presents. Even though those ambiguities were resolved in the works of Chernyaev's [45] and Nielson's et al. [53], a number of papers [50, 49, 52, 54, 43] offered variations of the algorithm which intended to improve it. Methods [31, 50, 49] use models with scalar fields, while [52, 54, 43] use binary models.

In the MC algorithm the surface extracted from voxels containing sharp model features is made up of chamfer triangles, so these features are lost.

Subsampling or decreasing the grid size provides more accurate results, but the problem is not really solved. If the model contains Hermite data (exact points and normals), Kobbelt et al. [50] developed a variant named *Extended Marching Cubes* that preserves features. It stores exact point intersections (of the grid edges and the surfaces) in grid vertices, and normals in grid edges. The intersection of the two or three faces (inside a cube) is approximated through a quadratic error function, and vertices and edges of the model are reconstructed inserting an adequate point in the cell.

The MC algorithm has no control over the shape of the extracted triangles and some of them may be poorly shaped. The method from Montani et al.

[52], the *Discretized Marching Cubes*, works on binary volume models and uses a triangulation scheme and lookup tables to generate a topologically correct surface using the values at the vertices of voxels. The exact position is replaced by the midpoint of grid edges, allowing integer arithmetic (except in normal calculations which are floating point arithmetic). Triangles obtained by the algorithm are stored in hash tables to facilitate later access. After the isosurface extraction step, a merging process eliminates unnecessary vertices. This process uses Freeman's chains [47] to find boundaries of coplanar polygons, eliminates unnecessary vertices and reconstructs vertices using lookup tables. Finally, normals of remaining vertices are computed.

In dual methods, points are not restricted to be placed at the edges of voxels. Nielson's method [54] computes a patch surface obtained from the intersections of three orthogonal planar curves (which lie on the edges of a grid). Then the centroids of these patches are linked as vertices of quadrilaterals, generating a dual surface. If a dual of the dual is computed, the resulting surface presents a smoothing effect.

Ju et al. [46] used a dual scheme to segment (contour) a signed grid of Hermite data in the edges. New vertices of voxels are computed using Quadric Error Metrics. For each edge stabbing the surface, the vertices of the four surrounding voxels are connected as in Surface Nets [48] -described in Section 2.2.4-. Since grids of voxels contain a majority of voxels without features, an octree is constructed from the grid in order to consume less memory. Also, the quadratic function error allows simplifications: the error of an interior node is computed as the sum of the errors of its eight children, and if this error is less than a given threshold, the children are pruned. Adjacency of leaves of different levels is arranged by adding triangles between the quads of the adjacent nodes. Finally, a heuristic is defined to test if the topology of a node and that of its subdivision is maintained. This heuristic can guide the construction of the octree depending on the user's desire to keep or not the initial topology of the model. The method is adaptable to segmented data. However, Dual Contouring methods can produce non-manifold topology in singular points.

Many alternatives provide algorithms using local decisions to solve MC ambiguities, but the algorithm described by Andújar et al. [43] provides a global strategy. Here a family of methods is proposed where a topological parameter or a combinatorial cost is optimized. The scheme identifies the situations which lead to ambiguities (X-faces and X-cubes) and creates data structures to store and manage their relationship. The Euler-Poincaré and other topological formulas provide the elements (number of triangles, handles and shells) to maximize or minimize in the algorithm.

The method proposed by Allamandri et al. [44] improves the precision of the reconstructed surface using a mesh refinement approach. A recursive

local refinement is done to construct an adaptive piecewise linear representation. Then, a geometric difference between each face and its ideal isosurface is computed and if it is greater than a given threshold, splitting points are added.

The work of Dietrich et al. [46] states that in MC, sampled points that are near grid vertices generate poor triangles. They propose moving the vertices (but keeping the topology of the grid and the isosurface) and apply the standard MC algorithm. Vertices can be moved in a parallel or orthogonal direction to the isosurface. Each method proves to be the best in certain situations, leading to the conclusion that an iterative combination of both improves the resulting mesh than any of them separately.

Manikandtan et al. [51] decreased the computation time and memory cost in spite of inaccuracies by only using one table and avoiding interpolation of vertex positions.

An adjacency lookup table was used by Wang et al. [56] to track connected surfaces instead of a cell-by-cell visiting strategy. The table guides which other cells are to be visited depending on the cell configuration.

An extensive survey, focused exclusively in the MC algorithm, was made by Newman and Yi [55].

Tetrahedral meshes

Some authors construct a tetrahedral mesh (volume) or a triangular mesh (surface) from the model. MC algorithms are ill-suited to deal with this distribution of data.

Marching Tetrahedron [29] is a MC-like algorithm, specific for tetrahedral volume subdivisions. It uniformly partitions space with tetrahedrons instead of cubes, even though this method produces more triangles than MC.

The method by Andújar et al. [28] extracts triangles in flat regions of the model and point sets in curved areas. It creates a discrete band of voxels around the isosurface which are decomposed in tetrahedra. For each facet the planarity of its neighborhood is computed and a radius is assigned to it. Regions are scanned in decreasing radius and tetrahedra in these regions are connected as tiles. Edges are generated from intersecting tiles. Tetrahedra not included in any tile are associated with an oriented point; its direction is the average of the three nearest faces.

Zhang et al. [40] extracted tetrahedral meshes from scalar points in grids. If the quality of data is not good enough, a denoising process is executed. The isosurface extraction uses Dual Contouring [49], generating a vertex inside each cell using quadratic error functions. It tetrahedrizes the boundary and interior cells in an adaptive octree, avoiding hanging nodes and cracks between neighbor leaves of different levels by applying heuristics. Correct topology is granted

by a recursive subdivision of the cube that contains a non-manifold dual contour. An error function approximates the difference between isosurfaces from neighbor levels: if there is a large change, it is assumed there is a feature and no simplification is made there. The method looks for poorly shaped tetrahedra and makes a contraction of edges to erase those tetrahedra. It uses three measures of every tetrahedron and, if one of these is below a threshold, the contraction is made.

The Cuberille method [59] is aimed to rendering since it does not guarantee the resulting surface to be manifold. If the polygon surface has to be manifold to apply geometry processing tools, other techniques must be used. Nielson [60] generalizes the Cuberille method to a tetrahedral volume and returns a manifold surface. Given the tetrahedral decomposition of a binary model, it finds the tetrahedron that intersects the surface (16 cases, in Cuberille's there are 23). An interior point (of those tetrahedra) is computed by minimizing the discrete norm curvature in an iterative process. These points become the mesh vertices of the surface.

Buatois et al. [58] designed an algorithm that uses the GPU to extract the isosurface from an unstructured tetrahedrical grid. It is based on Pascucci's method [61] but vertices are shared among adjacent tetrahedrons to avoid a bottleneck. Instead of sending the data through vertex arrays, Buatois uses textures to pass the data to the GPU.

A method that combines tetrahedra and MC is that of Scholz et al. [62]. The diamond hierarchical subdivision -a binary tree- of [63] is followed obtaining tetrahedra, which can be refined recursively in hexahedra depending on the viewer's position. MC is applied to resulting (non-regular) hexahedra.

A survey of extraction methods (especially those derived from MC, but not restricted to them) was presented by Andújar et al. [57]. Some previous definitions are made to characterize the methods depending on what kind of geometries they can reconstruct and whether they can control (locally or globally) the final topology.

2.2.4 Smoothing

Extracted surfaces often show flaws as staircase artifacts, noise -from measured data- or sharp edges when smooth surfaces were desired. Some authors propose methods to fair such surfaces.

Methods based on Gaussian smoothing suffer a shrinking effect. Taubin [74] used an iterative schema that applies two gaussian steps -one with a positive scale factor and then with a negative one-, which has a low pass filter effect (the curvature becomes the frequency and the artifacts correspond to high curvature noise). Desbrun et al. [69] extended the Taubin's work to irregular

meshes using geometric flows.

Surface Nets is an algorithm proposed by Gibson [48] that minimizes staircase artifacts in binary volume models. A node at the center of each cube is added and linked to its neighbors. Its position is iteratively modified to reduce an energy measure of the links (sum of squared lengths), and constrained to its original cube. The resulting surface is a smoother version of that of the MC, with the property that is consistent with in-out vertices. It is a dual method, since every vertex corresponds to an edge of the MC.

Chica et al. [68] have designed an algorithm to extract flat surfaces from a binary model, to smooth sharp features but preserve sharp edges. Sticks (edges of voxels that connect in-out vertices) of the model are identified. A flat is a cluster of sticks stabbed by a plane, and flat surfaces are chosen by a voting scheme. A boundary between flat and curved surfaces can be a sharp edge or an edge to be smoothed; the angle between normals is used to decide which kind of edge is. A smoothing displacement is computed using a bilaplacian operator averaging the displacements in two orthogonal directions. This displacement is clamped to the stick, and the process is iteratively repeated. Sharp edges are restored by identifying triangles that have edges belonging to two different faces (which can be flat or curved), subdividing these triangles and computing the correct edge.

Tasdizen et al. [75] proposed a diffusion-type smoothing on the normal field and then evolve the surface to match the new normals. They stated that Mean Curvature Flow is a good method to reduce noise in the model, but it also reduces its features. An intrinsic Laplacian of mean curvature flow is used to overcome this limitation.

The non-iterative method of Jones et al. [70] uses local first-order predictors and robust estimators of triangular meshes -that can be unstructured-. The new position of a vertex is calculated from predictions in its spatial neighborhood. Points in opposite sides of a feature are considered outliers and ignored, so the feature is preserved.

To remove noise from a surface, Sun et al. [73] iteratively filter noisy normals by weight averaging neighbor normals. Weights are set to preserve features of the model. Finally, vertex positions are iteratively moved to be adjusted to the new normals.

Wang et al. [76] proposed a surface fitting method for a variety of mesh models, even quadrilateral ones. They used vertex projection, curvature estimation via a quadratic model and a labeling scheme.

The extraction method from Lempitsky [71] uses convex quadratic formulas to obtain surfaces. It imposes higher-order smoothness rather than the minimal area property.

Zhong et al. [77] optimized an anisotropic triangular mesh surface by map-

ping it to a higher dimensional space where a sampling and optimization process is performed. Afterwards, they map it back to the original space.

Chen et al. [67] propose the use the parallel capabilities of the GPU to polygonize and optimize isosurface meshes from implicit surfaces. Their parallel version of the MC algorithm requires the explicit storage of one-ring neighborhood of each vertex. The MC ambiguity problem is not addressed here. The optimization part is performed with surface flows [72].

Some approaches to solve this problem, discussed in the following paragraphs, are based on a modification of the input data before the extraction.

Volumetric datasets often contain errors which can lead, during isosurface extraction, to a surface with more genus than the actual surface. If we know the topology of the actual surface and we find an initial estimate with the same topology, the algorithm of Botsch and Kobbelt [64] performs the morphing preserved topology. The preservation of topology is accomplished by maintaining the Euler characteristic between transformations. From an initial set of voxels, they are eroded and the holes filled. Finally, the model is inflated to reach the final structure.

Given a binary segmented 3D grid of medical data (each voxel belongs to one tissue), the method of Botsch and Kobbelt [65] repairs the grid to avoid inconsistent triangulations before a Marching Cubes algorithm is applied. The grid is iterated once and critical edges and vertices are detected. These edges and vertices represent non-manifold parts of the surface of the tissue. The voxels of these critical elements (together with their neighbors) are subdivided by 5x5x5 and a tissue is assigned to them to avoid the problem. The subdivision is not actually written to the data structure, but computed on the fly when necessary. Adding user constraints, the method can handle thin tissues for which the grid is too coarse.

The reconstruction process from point clouds can be enhanced with a previous feature topology extraction. Chica [66] created a visibility map of the grid and labels every cell with the number of complete k-rings visible from it. An edge stops visibility of cells near the edge. But cells at the edge have the visibility of two faces, so large visibility values can be used to detect edges or vertices. A threshold can identify features from faces. Edges and faces can be curved. The algorithm finally finds connections between edges and vertices to compute the topology graph.

2.3 Hierarchical Volume Representation

Hierarchies are data structures with many uses in computer graphics. They are used to represent a scene of multiple objects with an object hierarchy,

mainly to detect collisions [95]. But it is also a method to represent a single object, usually to visualize it. The space hierarchy represents the volume of that object.

Partitioning trees from Shumacker et al. [97] was the first attempt in hierarchical representation of objects, with the hidden surface removal problem in mind like Binary Space Partition (BSP) trees from Fuchs et al. [87, 88]. But the BSP tree soon showed its strengths in other areas such as ray tracing [80], collision detection -in the *Quake* video game [93]-, shadow generation [85], spatial databases in GIS [81], robot motion planning [90] or image registration [84] among many others, including solid modeling [92]. Even CSG operations can be performed using a BSP implementation, as Bernstein and Fussel [83] proved.

BSP trees have been the starting point of other hierarchical structures, depending on what splitting plane is chosen and in which place is applied. One of the most used are k-d trees, created by Bentley [82]. They were devised for point range queries in k dimensional spaces, but it is also used in other circumstances like ray tracing [98] or photon mapping [89]. Addressing these situations in which the k-d tree is a critical data structure, Zhou et al. [99] propose a method to construct a k-d tree in real time on the GPU.

Like the BSP, the Bounding Sphere Hierarchy is used to represent complex scenes [79] or to model single objects [94]. As stated by Agarwal et al. [78], the hierarchy can be *layered* when a sphere in a parent node encloses the spheres of its children, and *wrapped* when a sphere encloses (tightly) the geometry.

Moreover, hierarchical volumes store in an implicit way a level of detail (LOD) representation of the object, as we will see in Section 2.3.2.

Octrees and their 2D equivalents, quadtrees -which are older [91, 86]-, are a hierarchical space partition with three simultaneous planes (axis-aligned) applied in each subdivision, resulting in 2^3 children, named octants. Due to its important use in volume modeling, they are reviewed in the next Section.

There are many more hierarchical schemes than those detailed in this Section. For an extensive survey, see Samet [96].

2.3.1 Octrees

An octree is a hierarchical representation of a parallelepiped region of space. The volume is subdivided in eight uniform octants, halving its boundaries by three mutually perpendicular planes. The octants have the same volume and are non-intersecting, defining a partition of space. Each node spans exactly in eight children, each one describing the space of a particular octant.

Octrees were first designed by Meagher [114] and since then, a number of uses have been given to this data structure. Selected examples are reviewed to

show such diversity.

Volumetric data in a scalar field can be represented by Min-Max octrees -every node stores the minimum and maximum value of its children- in order to efficiently extract isosurfaces (i.e. with a marching cubes algorithm) given a threshold value [119]. Mora et al. [115] used parametric cubes (which includes the 27 voxel neighborhood) to calculate a two-order filter. Values of the parametric cubes are stored in a Min-Max octree that allows the user to interactively change the threshold of the isosurface.

Ordered traversing can lead to rendering strategies. Livnat and Hanset [111] traversed the octree in a view-dependent way to cull nodes occluded by previous ones.

Three dimensional texture rendering was introduced by Cabral et al. [103] and Wilson et al. [120]. In the work of Boada et al. [102] a large 3D texture block is decomposed and stored in an octree. The octree is traversed driven by data importance and the selected nodes determine the decomposition of the texture.

Laine and Larras [113] defined an out-of-core sparse voxel octree (SVO) and use it for rendering with ray-casting. Baert et al. [101] emphasized the construction phase of the SVO -from a voxelization of a massive polygon mesh- and then stored it on disk using a Morton ordering which eases later accesses.

Lefebvre et al. [112] presented a method to store an octree in a texture and thus the possibility to be processed by the GPU. Pharr and Fernando [116] used a 3D texture to store the indirection pool in the GPU, which are 2^3 cubes -indirection grids- with data or addresses to other indirections grids.

Losasso et al. [110] simulated water and smoke with an octree that allows an adaptive mesh strategy. Ferstl et al. [104] also model dynamic fluids: the original hexahedral grid is coarsened into an octree and conclude that it results in very little differences in the simulation.

Ju [108] repairs a polygonal model embedding it in an octree, defines intersection edges and then reconstructs the surface with dual contouring. Bischoff et al. [35] used an octree to store references to the triangles that intersect it. It refines in regions of high complexity or boundaries and then topology is restored using an extension of the dual contouring algorithm. The number of output triangles is too high and a standard mesh decimation algorithm is applied.

Octrees have also been used as object representations in ray tracing algorithms. The main issue in this situation is the efficient traversal of the octree the ray must perform. Glassner [105] made a top-down search. Samet [117] used a bottom-up approach with a lookup table to compute the next node depending on the direction of the ray. Sung [118] used a 3DDDA algorithm to locate the next hi-res cell, whether it was a real node or virtual; an oc-

tree is stored using a hash table lookup. Knoll et al. [109] constructed an octree from a large volume, which can be rendered efficiently with a min-max implementation and hashing.

There has been some effort in sculpting procedures. In one of them, Bærentzen [100] used octrees as a basis, and the editing process was made with a CSG approach and adding or subtracting tools.

Hornung et al. [106] introduced the *OctoMap*, the real-time octree representation of a volumetric environment captured by robot navigation. The octree distinguishes occupied space from free and unknown areas. And Jessup et al. [107] propose a method to merge occupancy 3D octree maps, scanned with a team of independent robots in large environments.

2.3.2 Multiresolution Models

In several circumstances a model with a great level of detail become a disadvantage to the program that visualizes it. If, for instance, features of the model are to be rasterized in less than one pixel there is no need to reach such precision. Indeed, it is a drawback since too much geometry is sent to the pipeline. These situations can arise when a metric exceeds a given threshold, as when the model or a part of it is far away from the viewer, or some objects do not intersect a region of interest.

There are a number of algorithms that simplify a given model in order to obtain less geometry. Thus, we can have several geometries -each one of them named *level of detail* (LOD)- of any single object of the region, ready to be rendered when necessary.

The multiresolution method constructs a tree of the scene or volume, where every node contains a simplification of the geometry. Variable level of detail representations of parts of the model are obtained by defining a cut of the hierarchy (the *front*). Some methods [129, 124] render the leaf nodes of the front; others [128] merge all nodes above the front.

Designing algorithms that obtain LODs that are rendered with no appreciable loss of visual quality remains a topic of interest.

Multiresolution trees and simplification algorithms have been studied by many authors.

Cignoni et al. [124] proposed a technique dubbed *Adaptive TetraPuzzles* that organizes the data in a multiresolution tetrahedrical structure. Lindstrom et al. [136] stated the desirable criteria that real-time multiresolution systems should include: efficiently queryable mesh geometry; dynamic changes to the geometry should not significantly impact the performance of the system; localized high frequency data should not have a global effect on the complexity of the model; small changes to the view parameters should not drastically

change frame rates; and the loss in image quality should be bounded as input parameter. Although the distance of an object from the point is the simplest way to choose the level at which this object will be displayed, Funkhouser and Séquin [126] were the first to apply LOD to maintain the frame rate at an interactive rate. Andújar et al. [121] proposed a method to obtain different simplifications of the original model in number of elements and topology. In a following work [122] the simplification process preserves features and color using edge collapsing. Discretized Marching Cubes [52] is applied to preserve edges and vertices not previously aggregated. Using the progressive mesh [132] as a base, Hu et al. [133] produced a LOD mesh refinement that operates in parallel entirely in the GPU.

Multiresolution techniques for volume models [141, 142] use specific data structures. Initial schemes such as Boada et al. [102] and Lamar et al. [135] were based on textured octrees, while more recent proposals like the one from Gobbetti et al. [130] uses block-based hierarchical data structures. Space subdivisions with a large number of children per node, with N^3-Trees and mipmapped textures have been used in the Gigavoxels scheme [125] from Crassin et al., while the approach from Guthe et al. was based on volume Wavelets [131]. Other approaches include 3D mipmaps on hierarchical grids [127] and sparse/compressed representations [140, 144]. Wang et al. [143] concentrated on the preservation of features in data reduction. This is done by assigning importance to voxels according to the current transfer function; in our case, we do not change the reduced data, but we adapt the TF to better provide the information on the original data. Kraus and Bürger [134] focused on the interpolation and reduction of RGBA data.

The *Far Voxels* approach [129] of Gobetti and Marton is an interesting multiresolution approach to surface data and volume representations, supporting heterogeneous surface models. It incorporates visibility culling, out-of-core construction and level of detail. An axis-aligned BSP is generated: the leaf nodes contain chunks of a maximum number of triangles, which is stored off-core; and inner nodes are arrays of voxels. Each voxel contains an approximation of that part of the model viewed from different points of view. If the projected voxel size of a node is below a threshold, it is rendered as a splatted point. If not, the algorithm descends one level. When it reaches a leaf, the detailed geometry is rendered. The algorithm also includes optimization with occlusion tests, data structures for asynchronous I/O requests and RAM and GPU caching.

2.4 Compression

Compression techniques consist in decreasing the size of volumetric data by applying a specific encoding algorithm [146]. In some approaches compression is combined with a brick or a hierarchical partition of the original volume to facilitate the compression process, and a level-of-detail rendering/transmission. The compact representation can be lossless or lossy. Medical applications often require a lossless compression when rendering models, even at coarser resolutions.

Vector quantification is a lossy compression method [152]. The volume dataset is represented as indexes into a small codebook of representative blocks. This scheme is suitable for a CPU-based ray-casting render. Schneider and Westermann [154] proposed a Laplacian pyramid vector quantification approach that allows relatively fast volume decompression and render on the GPU. However, this method does not support GPU linear filtering capabilities and its rendering cost increases when using high zoom factors.

Wavelet transforms offer considerable compression ratios in homogeneous regions of a volume while conserving detail in the non-uniform ones [151, 147]. Ihm and Park [150] proposed an effective 3D 16^3 block based compression/decompression wavelet scheme. Guthe et al. [149] proposed an alternative hierarchical wavelet representation which allows to store very large volume datasets in compact but lossy data structures.

Faut and Kwan-Liu Ma [148] proposed a method which uses volumetric compression and decompression plus rendering on the GPU for large datasets. This method is based on a block classification scheme that allows real-time rendering.

Suter et al. [153] presented a hierarchical tensor approximation volume visualization approach. They created a new global TA hierarchy for multiresolution DVR and multiscale feature visualization.

In [145] Agus et al. implemented a method for simplifying blocks of voxels from segmented volume data, providing a multiresolution scheme. Their method allows interactive manipulation of the transfer function and a GPU ray-casting framework.

2.5 Surfel Representations

From the early moment in which Catmull [157] noted that recursive subdivision of a model can lead to a single point, points became an alternative to triangles as rendering elements. They have long been used to render objects that have difficult geometry, like any kind of fluid such as smoke, fire, clouds or explosions,

or from models with no topology, like scanned data. They are also useful when the contribution of any rendering element is less than a pixel.

Point rendering needs to define the sampling frequency of the modeling stage. Alexa et al. [155] resampled the possibly noisy or redundant point set to obtain a surface with a given accuracy, using local moving least-squares. Following their approach, Guennebaud and Gross [159] fitted a sphere surface rather than a plane, obtaining a tighter fit and more stability in low sampling frequencies.

Pfister et al. [164] proposed the use of surface elements, or Surfels, to render complex geometric objects with point primitives and without topology information. A surfel is an element with shade attributes that locally approximates a surface. Surfels are usually represented as small disks, including a 3D point (the disk center), the disk radius and the normal vector.

The Elliptical Weighted Average (EWA) [160] is a space-variant filter that computes a weighted average over an arbitrarily oriented elliptical area. It produces a very high quality anti-aliasing, but it was not supported by the GPU. Several enhancements have been proposed [162, 167, 161, 163], many of them take advantage of the GPU to perform the EWA.

Adams and Dutré [156] proposed an interesting method in which two objects, defined by surfels in octrees, perform CSG-style boolean operations. Surfels in intersections overshoot each other and are resampled in a lesser size to better adjust to the intersection edge.

Range scanners are devices that capture the geometry of a static or dynamic 3D scene. The result is the series of sampled 3D points, or *point clouds*, with no topology information. Surfels are a good method to display such kind of data.

Stückler and Behnke [165] devised a method to reconstruct an indoor scene with a moving RGB-D camera. Color and shape distributions in multiple resolutions are stored in octrees and are represented with surfels.

Carceroni and Kutulakos [158] addressed dynamic 3D scenes with *dynamic surfels*, which include movement information, but in a small range.

Slomp et al. [166] have captured moving objects with high speed cameras. Color of points is used to store normals, hence the whole image is monocolor. Surfels stored in hierarchies are the elements used to render the image.

2.6 Client-Server Architectures

Client-server architectures and progressive transmission algorithms started almost two decades ago with mesh-specific methods. Hoppe [132] proposed an algorithm for the transmission of triangle meshes as a sequence of progressive

smaller meshes.

The previously commented work of Gobetti and Marton [128] is well-suited for client-server architectures and huge point data clouds. In their approach each node of a k-d tree hierarchical data structure contains a region of space with a similar amount of equally distributed samples that are complemented by the samples in upper levels. Rendering traverses the tree from the top and continues until a predefined level/density is reached.

P. Callahan et al. [168] implemented a client-server architecture for progressive volume rendering of unstructured data. They used the server as a data repository and clients as render machines that accumulated the incoming geometry and displayed it in a progressively way. The server processes the model to create an octree, which is traversed by depth ranges from front to back. When the client requests a certain packet, the server culls the geometry outside the current depth range and send the visible volumes to the client, also incrementing depth ranges for future steps.

Luke and Hansen [170] classified the client-server architectures in four scenarios, from *thin-client* -where the client is only responsible for displaying the images sent from the server- to *fat-client* -where the server is only data storage and all calculations and rendering are performed in the client-. They also proposed a method where the rendering load is divided between client and server. From a given viewpoint, depth buffer and color buffer are the basis to create a triangle mesh of visible data. The mesh is sent and the client renders it, having the possibility to perform some variations on the viewing parameters.

Zhou et al. [171] presented a client-oblivious model due to the great diversity of mobile devices and wireless networks. Depending on the client rendering power, it renders points, surfaces or volumes. The server generates a unique representation of the model (an occupancy image). From it, which can be easily rendered as points, the isosurface and volume can be reconstructed and rendered. The structure is hierarchical and allows progressive transmission.

Gobetti et al. [169] have proposed an algorithm for rendering gigantic volume models. In their client-server approach the volume is decomposed into a multiresolution hierarchy of bricks. Each brick is further subdivided into smaller blocks, which are compactly described by sparse linear combinations of prototype blocks stored in an overcomplete dictionary. The dictionary is learned, using limited computational and memory resources, by applying the K-SVD algorithm to a re-weighted non-uniformly sampled subset of the input volume.

For a more complete survey on volume rendering architectures, client-server schemes and compression techniques, we refer to the work of Balsa et al. [146].

2.7 Low-cost Portable Client Devices

Hospitals are more and more interested in tele-medicine and tele-diagnosis. Client-server applications support these functionalities. Sometimes the use of mobile and low-end devices is necessary due to their portability and easy maintenance. However, low performance hardware properties make it quite complex to build efficient visualization systems on these devices.

Nowadays, the majority of portable devices include a GPU capable of running OpenGL ES and other API. See the state of the art from Capin et al. [173].

Lamberti et al. [176] implemented a technique in which storage and computation resources are provided by a server system while mobile devices are only used as client front end. User interaction in the mobile device is encoded and sent to the server which, in turn, produces the corresponding image in its frame buffer. The image is sent via TCP to the client which shows it in its display. With low bandwidth, the image is compressed to jpeg or gzip, but requires time to decode in the client part that decreases the frame rate. The image can also be sent in half resolution when the model is being rotated.

Moser and Weikpof [178] introduced an interactive technique for volume rendering on mobile devices that adopts the 2D texture slicing approach.

Noguera et al. [179] proposed an algorithm that overcomes the 3D texture limitation of mobile devices and achieves interactive frame rates by caching the geometry of the slices in a vertex buffer object.

The method proposed by Díaz et al. [174] produces expressive medical illustrations. Segmented medical datasets can be cut by a clipping plane defined by the user and then selected organs can be extruded out of the plane by dragging them with the mouse.

An interesting application for mobile devices can be found in [180]. The authors analyzed current graphics hardware in most high-end Android mobile devices and performed a practical comparison of volume rendering, as a well-known GPU-intensive task. The paper discusses implementations of three different classical algorithms, to show how the current state-of-the art mobile GPUs behave in volume rendering.

Campoalegre et al. [172] proposed a visualization scheme based in equal-sized blocks which are compressed via wavelets and sent to the client device, which only decompresses the blocks in the region of interest.

ImageVis3D, a program based in the previously revised Tuvok architecture [175] has its mobile version from the same authors. Due to the adaptability of the method to different GPUs, the program keeps an interactive frame rate visualizing large datasets, even in mobile devices.

Last but not least, Movania and Feng [177] presented a single-pass ray

caster and a 3D texture slicer rendering algorithm for the WebGL platform, capable of handling dynamic transfer functions in mobile devices.

2.8 Perceived error in images

CGI applications often are in the need to compare images produced from different methods to determine their validity or efficacy. Usually, the human eye perception is not the most effective tool to perform such comparison and more formal procedures are required to do it.

In the pre-digital era, there were a number of methods to describe the quality of images. They were mainly based in general parameters of the photography, like the Modular Transfer Function of the system or the noise power spectra, but these methods were not well suited for digital images. For this reason, new methods based on pixel processing appeared.

In this context, the visible differences predictor algorithm (VDP) from Daly [181] tries to, beyond the physical differences between images, describe the probability that these differences become visible to the human perception. VDP computes an absolute value of the quality of an image, and presents an image with the differences between two previous images, seen by an average human observer. The algorithm considers several physical elements of the visual and neural system. The resulting image places a probability value in every pixel in the context of the reference image, to facilitate the interpretation of the result. Mantiuk et al. proposed an extension of VDP to encompass high dynamic range (HDR) images [182] and [183], being an extensively used algorithm.

2.9 Conclusions

A modern anatomical Atlas is a complex work that includes many areas of computer graphics. It involves implementation challenges that currently are not fully solved, or not solved in this particular field. Specific conclusions for Atlas design can be derived from the previous work presented in this Chapter.

In anatomical Atlases the primary goal is the inspection of structure boundaries. Therefore, an indirect volume rendering algorithms are well suited. The design of a scheme of isosurface extraction relies on the desired properties of the resulting surface. Since Atlases are intended for visualization only, not for edition, it is not essential to generate the topology of the model. Surfels are a suitable representation for models without topology.

Octrees offer a convenient storage for surfels. With the exception of geometric location inherited from parent to children, the information contained in a node is independent and not connected to remaining nodes. Consequently, there is no restriction in deciding which surfels to render. Selective traversals allow a LOD visualization of the model, or to give emphasis to a region by rendering it at a higher resolution than neighbor regions.

Low-cost portable devices present an added challenge since the reduced strength and resources of these machines compels to a very accurate design of applications. Otherwise, any weakness in data structures or algorithms will bring the inefficiency and lack of usability of the result.

The client-server architecture is a widely used scheme when a given device is not powerful enough to execute certain applications. The server can perform calculations and/or provide data on request, depending on the features of the client. Nowadays the processing and rendering power of low-cost devices is increasing, leading to a fat-client approach where the server only supplies preprocessed data.

Data compression is an essential task to be accomplished in order to make the Atlas work properly in low-cost portable devices when a client-server architecture is used. Generally, compression methods are biased to the specifics of the data they use, so new schemes are usually conceived for every new project. Atlases contain big amounts of information to be stored and transmitted, so a good compression method for both, octree and surfels, has to be devised.

As has been presented in this Chapter, there have been proposals directly or indirectly related to Atlases that contribute to a final, complete application. But there is enough room to find new solutions and new ways to integrate those proposals, especially in hierarchical representations. This is the goal of the next Chapters.

Chapter 3

Surfels in Space Subdivisions

The main concern of users of Medical Atlases is the visualization of organ boundaries. Therefore, surface representation methods should be researched. We investigate schemes with independent local geometric primitives, since they do no require topology to be included and lighter models can be generated. Point-based methods and Surfels fulfill those conditions. Our approach proposes a specific surfel scheme that uses a hierarchical spatial subdivision -taking advantage of the volumetric nature of the model- to complement the geometry of the surfels. Thus, associating the surfels with the hierarchical structure, less information is needed to store the geometry of the surfel.

A method based on tetrahedra and scalar fields to encode the plane of the surfel is presented, resulting in a faithful representation of the planes.

Along with the geometry of the Surfel, shading information should also be encoded. Different color schemes for the surfels are analyzed and a realistic and memory efficient method is proposed.

3.1 Constrained Surfels

The chosen representation for the organ boundaries are Surfels. A *Surfel* is a disk with specific geometric parameters and material information for rendering purposes. The geometry is defined by the position of the disk center, its radius, and the orientation of the disk's supporting plane -represented by the normal vector-. Surfels where introduced by Pfister et al. [164] as rendering primitives without topology information that locally approximate a surface.

In this work, we propose the use of *Constrained Surfels*. This new definition leads to a compact representation of its geometry.

Let us assume that a spatial subdivision scheme is applied in the space occupied by the model. The subdivision produces a number of cells c_i, $i = 1..N$.

The union of all the cells is the Universe that contains the model, while the intersection of any pair of cells c_i and c_j, $i \neq j$ has a zero volume. As it is well known, these are the two properties that define a *partition* of the Universe. There are some types of spatial subdivisions: non hierarchical and uniform (model of voxels), non hierarchical and not uniform (tetrahedrizations), and hierarchical subdivisions (e.g. octrees). In hierarchical subdivisions, any cell in a level is the union of specific cells of the next level. Hence, every level becomes a partition of the Universe.

Given a spatial partition P of the Universe and its cell decomposition c_i, $i = 1..N$, a model of Constrained Surfels associated to P is a set of Surfels s_k with $k = 1..M$, corresponding to a subset S of cells $c_{i1}..c_{iM}$ of P, with $M \leq N$, so that every s_k corresponds to a certain cell c_{ik} and splits it.

Constrained Surfels have the interesting property that their radius and position becomes implicitly determined by the geometry of their corresponding cell. Accordingly to this, only the plane equation has to be stored in the cell. The center of the surfel can then be computed as:

$$\text{Center}(s_k)=\text{Centroid}(\ c_{ik} \cap \text{Plane}(s_k)\ )$$

While the surfel radius must simply ensure that the cell is completely split in two parts.

As a first example, let us suppose that P is a voxelization of a volume model V and that S is the set of voxels that contain a certain isosurface of V. Then, all cells in S will contain sticks with appropriate supporting planes that locally approximate the isosurface. To compute the stick centerpoint, an efficient and reasonable approximation consists in computing the centroid of the intersections between the stick plane and the cell edges:

$$\text{Center}(s_k)=\text{Centroid}(\ \text{Edge}_1(s_k) \cap \text{Plane}(s_k), \ldots\ \text{Edge}_n(s_k) \cap \text{Plane}(s_k)\ )$$

As a second example, P can be any space partition defined by a hierarchical space subdivision. In this case, cells corresponding to any tree node (at any level in the hierarchy) can become part of P and can therefore contain surfels.

3.2 Surfel Plane representations

In most cases, the efficiency of the algorithms is strongly related to the correct choice of the data representations. Choosing a suitable and compact plane representation is essential for constrained surfels, as planes are their only geometric structure. As planes are always represented through several numerical values or parameters, in what follows we will use the term *Plane Parameterization* to indicate the set of -usually integer- values that represent planes.

Our approach requires the parameterization to be faithful: the plane representation should not give more importance to some regions of the Universe Cube in detriment of others, and a regular grid in the parameterization space should map onto a uniformly distributed family of planes in 3D space. Also, it is useful to have linear parameterization, that is, point-plane tests should only involve linear operations on the plane parameters.

Moreover, in our case we will use constrained surfels in the framework of a multiresolution octree data representation (Chapter 4). We can therefore benefit from well known properties of multiresolution models in view-dependent visualization algorithms (like FarVoxels [129] and TetraPuzzles [124]). In particular, perceptual rendering criteria lead to a limitation on the projected size of the octree nodes (voxels) in the viewport, the consequence being that plane parameters can be discretized. In fact, any valid discretization should result in projected geometric errors of the surfels of the order of a pixel. Finally, we would like to have local plane parameterizations based on the geometry of the cubic cell of each surfel. Having these local plane parameterizations has two advantages:

- Local frames that support plane parameterizations are implicit in the hierarchical space subdivision and have null memory requirements.
- Having small domains (of the order of the surfel cell), plane parameterizations can be discretized in a smaller number of bits as the only discretization requirement is that it should lead to projected geometric errors of the order of a pixel.

In Section 3.2.1 some existing parameterizations are reviewed. Sections from 3.2.2 to 3.2.5 are devoted to CCPP, a new plane parameterization that is well suited for Constrained Surfels and meets the requirements just mentioned.

3.2.1 Plane Parameterizations

In this Section we present a non-exhaustive list of plane parameterizations. We will show that none of them can be adapted satisfactorily to Constrained Surfels. In Sections 3.2.2 and 3.2.3 a new proposal will be made, which was inspired by the main ideas from Andújar et al. [184].

Hough [189] presented a method to detect lines and arcs in photographs of subatomic particles. He exploited the duality point-line, where points reside in the $\mathbb{R}^2$ space and lines lie in a parametrized 2D space slope-intercept (see Figure 3.1). Scanning of the parameterized space leads to the recognition of desired features.

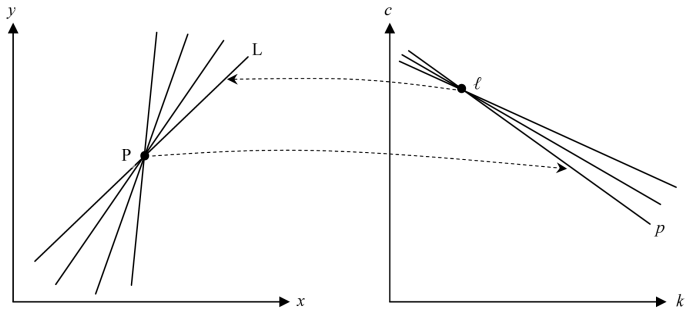


Figure 3.1: Line-point duality from Hough [189]. Line L in $\mathbb{R}^2$ space satisfies the equation $y = kx + c$, which corresponds to the point l in the parametrized slope-intercept space. And line p in that space corresponds to point P in $\mathbb{R}^2$. Image from [185].

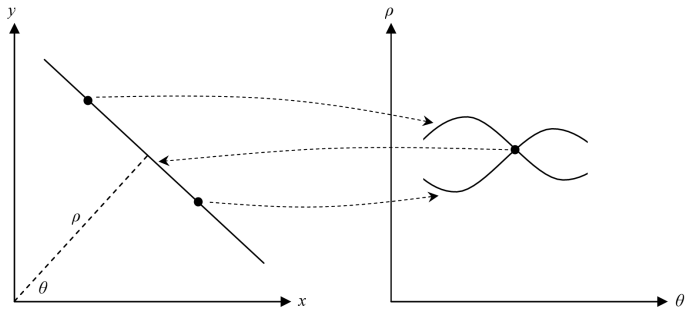


Figure 3.2: Normal form parameterization [187]. Points in $\mathbb{R}^2$ become lines in the parameterization and their intersection -a point- corresponds to a straight line. Image from [185].

Duda and Hart [187] observes that the Hough parameterization slope-intercept is unbounded in vertical lines, which have an infinite slope. They introduce the *normal form* parameterization: $\rho = xcos\theta + ysin\theta$, ρ being the minimum distance from the origin to the line, and θ the inclination of this distance. θ is restricted to $[0, \pi)$ so the parameters of a given line are unique. In the (ρ, θ) space a line in $\mathbb{R}^2$ becomes the intersection of curved lines, as can be seen in Figure 3.2. Many generalizations of this parameterization have been made since then.

The normal form of a line in $\mathbb{R}^2$ can be extended to planes and $\mathbb{R}^3$ using spherical coordinates (θ, ϕ) for the normal direction and ρ for the distance, as Décoret et al.[188] propose. They assume that in polar zones the parameterization is not uniform, oversampling being required there.

However, these three parameterizations (Hough, Duda, Decoret) are not linear, and tests between 3D points and planes are more computationally expensive than in the case of linear schemes.

Andújar et al. [184] define a parameterization of a plane inside a cube. They state that the plane must intersect the cube in 3, 4, 5 or 6 edges which, considering rotations and symmetries, results in the six cases. They also consider that the plane can be described uniquely with only three of these edges. If the edges are carefully chosen, the six cases are reduced to four. Hence, the plane can be parametrized with an integer that identifies the intersection case, the stabbing point in the three edges (3 reals which can be quantized) and a bit for the orientation of the plane. It is easy to verify that this parameterization is not linear.

An obvious linear parameterization is the classical representation by the four floats of the implicit equation $a \cdot x + b \cdot y + c \cdot z + d = 0$. However, using four float values is too verbose in most cases. An alternative is to encode the normal vector of the plane plus the d value. Meyer et al. [190] discuss a number of alternatives for normal vector encoding in this case, including the SC encoding with spherical coordinates in a similar way as [188], the cube map CC that encodes the direction by its corresponding point in the boundary of a cube centered in the origin, the parallel projection PP, the sextant parallel projection SPP, or the octahedron normal vectors ONV.

3.2.2 Tetrahedron-Based Parameterization

As already mentioned, suitable plane parameterizations for constrained surfels should fulfill, if possible, the following properties:

- The plane parameterization should be local, being based on the geometry of the cubic voxel of each surfel.
- The plane parameterization should be linear, that is, the constraint that a point (x, y, z) is inside (or outside) the half-space defined by a plane is represented by a linear constraint in the parameterization space.
- The plane parameterization should be faithful, that is, the plane representation should not give more importance to some regions of the Universe Cube in detriment of others, and a regular grid in the parameterization space should map onto a uniformly distributed family of planes in 3D space.

Let us assume that we have four points in space, defining a non-degenerate tetrahedron (see Figure 3.3). Then, any oriented plane can be represented as

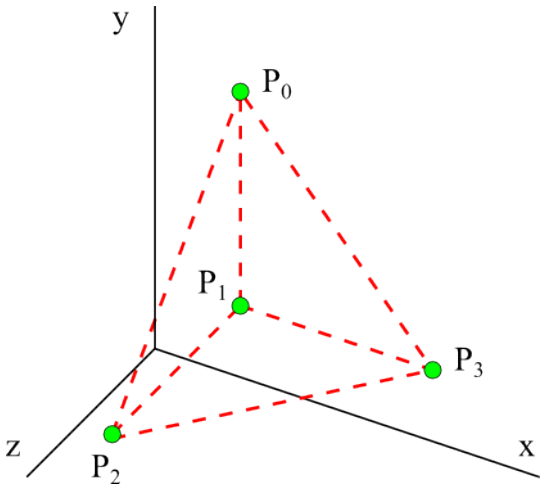


Figure 3.3: A tetrahedron defined by its four vertices $P_0..P_3$.

a tuple of four scalar values (weights) at the points $P_0..P_3$: (w_0, w_1, w_2, w_3). These weights can be interpreted as the values defining a scalar linear field, the plane being its zero-isosurface. They can also be interpreted as the signed distances from the tetrahedron vertices to the plane. There is a degenerate case where all weights have the same value: in this case, the field is constant and the zero iso-surface is the plane at infinity. For any 3D point Q, the value of the linear scalar field at Q can be easily computed as the linear combination of the weights times the barycentric coordinates of Q with respect to the tetrahedron vertices.

3.2.3 The Connected-Cubes Plane Parameterization

The plane parameterization based on the weights at the vertices of a tetrahedron is redundant: each plane is represented by a straight line in the 4D space of the values $w_0..w_3$. Let us first consider a 2D example. In this case, the base tetrahedron is a triangle like the one represented in Figure 3.4.

In this case, hyperplanes in 2D (straight lines) can be represented by points in the 3D space of the weights $(w_0..w_3)$. But since the tuple $(\lambda w_0..\lambda w_3)$ defines the same plane for any value of λ, planes are represented by straight rays from the origin, see Figure 3.5.

We choose a lower dimension representation for the planes in order to have a unique point representing each oriented plane. Planes are represented in this

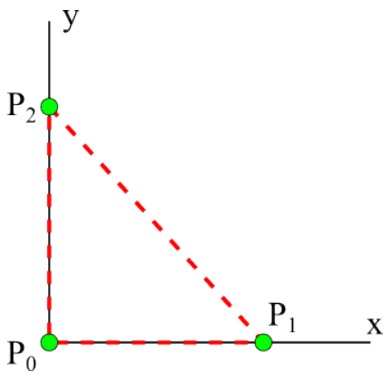


Figure 3.4: A 2D example: the triangle is defined by its vertices P_0, P_1, P_2.

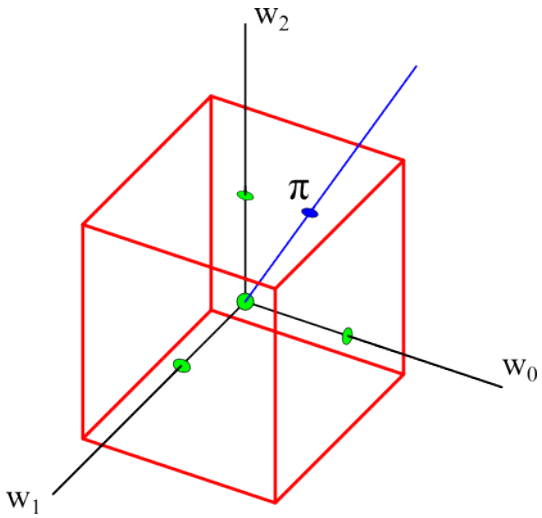


Figure 3.5: Any point in the blue line defines the same plane. Planes can be represented by the intersection point π between this line and a face of the Cube centered at the origin.

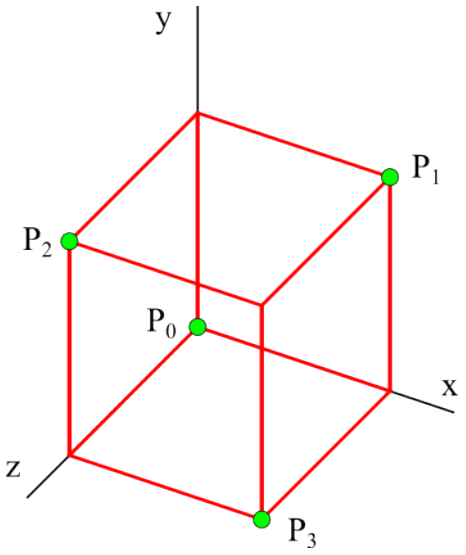


Figure 3.6: The four vertices of the base tetrahedron for the Connected-Cubes Plane Parameterization.

case by the intersection of the corresponding line -blue in the Figure- with the faces of a cube centered at the origin in the parametric space $w_0..w_2$.

Returning to 3D, let us now assume that our base tetrahedron is defined by four alternate vertices of the Universe that contains the modeled scene, see Figure 3.6.

Any oriented plane is represented by a line through the origin in the four-dimensional $w_0..w_3$ space. The Connected-Cubes plane parameterization represents it by the intersection between this line and the 3D faces of a hypercube centered at the origin. This is a set of eight 3D cubes, corresponding to the maximum and minimum values of the four weights. If we assume, without loss of generality, that the hypercube is defined by $|w_k| \leq 1$ for all k, any oriented plane is unambiguously defined by one point in one of the eight C-cubes that will be noted as $w_0^+, w_0^-, ...w_3^+, w_3^-$. The cube w_k^+ is the face of the hypercube corresponding to $w_k = 1$, whereas the cube w_k^- is the face of the hypercube corresponding to $w_k = -1$:

$$w_0^+ : \{w_0 = 1,\ -1 \leq w_k \leq 1,\ k = 1, 2, 3\}$$

$$\cdots$$

$$w_3^+ : \{w_3 = 1,\ -1 \leq w_k \leq 1,\ k = 0, 1, 2\}$$
$$w_3^- : \{w_3 = -1,\ -1 \leq w_k \leq 1,\ k = 0, 1, 2\}$$

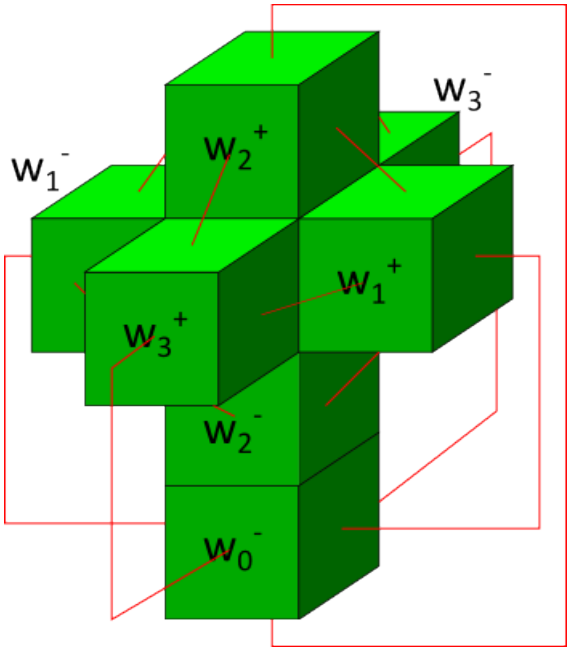


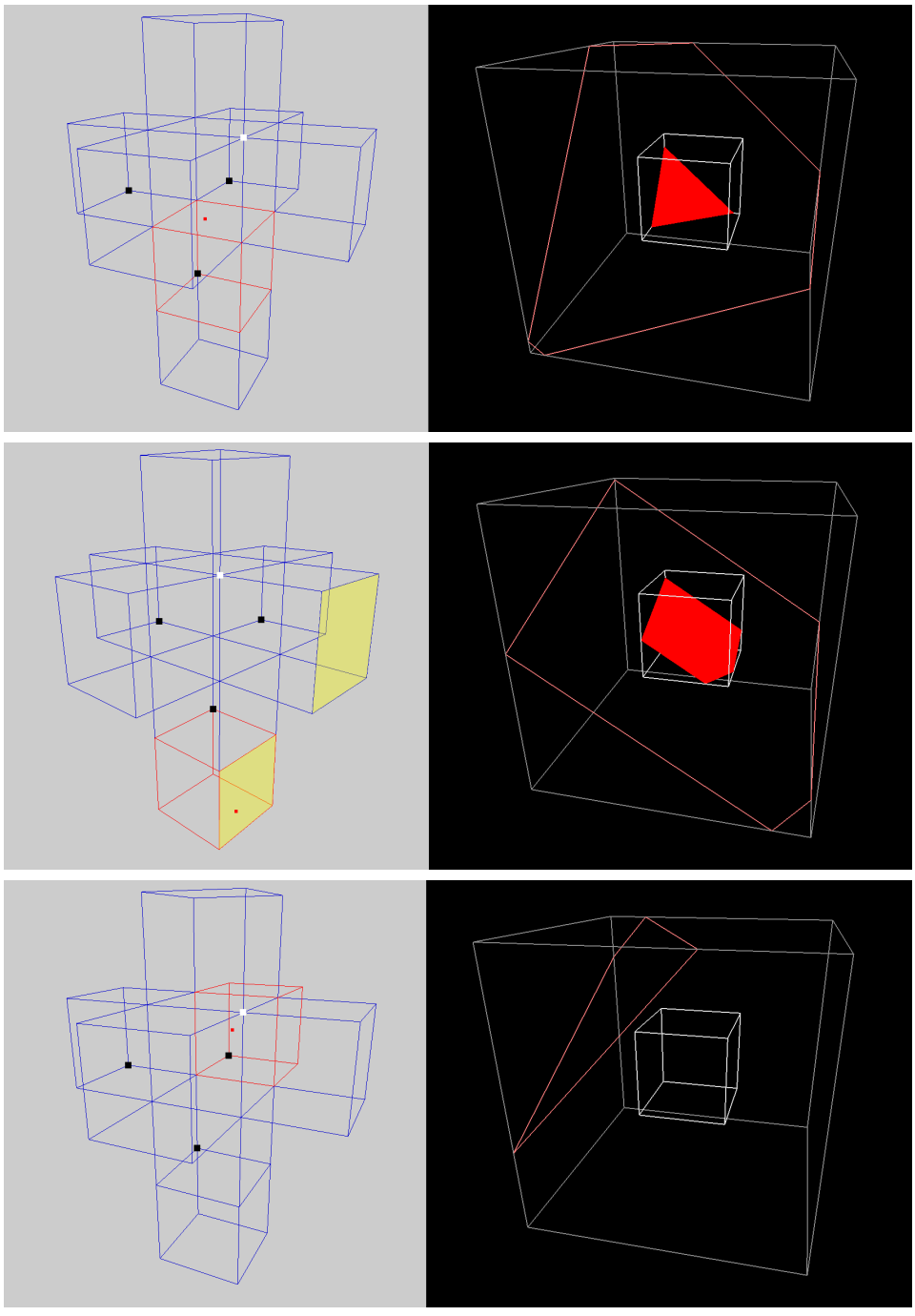
Figure 3.7: The eight cubes of the Connected-Cubes Plane Parameterization and their connections. The w_0^+ is the central and occluded cube. Connections between faces have been represented with red lines. Adjacent faces are also connected.

The eight cubes of the Connected-Cubes plane parameterization are topologically face connected. They represent a boundaryless 3D manifold -the boundary of the hypercube- and each of them has six face-connected neighbors, see Figure 3.7. This boundaryless 3D manifold is formed by 8 cubes, 24 faces (or 48 half-faces), 32 edges and 16 vertices. The 24 faces correspond to all possible combinations $(w_k^+, w_l^+), (w_k^+, w_l^-), (w_k^-, w_l^-)$ with $k \neq l$. Its 16 vertices show all possible combinations $w_k = \pm 1 : (-1,-1,-1,-1)..(1,1,1,1)$.

Each of the eight cubes has one singular vertex. It corresponds to $w_0 = .. = w_3 = -1$ for the w_k^- cubes, and to $w_0 = .. = w_3 = 1$ for the w_k^+ cubes. They are the two intersections of the line $w_0 = .. = w_3 = 1$ with the boundary of the hypercube. They correspond to a constant scalar field, its zero-isosurface being the plane at infinity.

For any of the cubes, the three faces that do not contain the singular vertex will we noted as the significant faces of the cube from now on.

In Figure 3.8 a plane and its corresponding parametric point are shown. At right, the plane is drawn along with the Universe (the small cube). Not



a)

b)

c)

Figure 3.8: Plane and its parametric point. At the right column, the plane in the 3D space. At the left column, the corresponding parametric point (in red) in the unfolding of the 4D parametric space. In white and black, the two singular points.

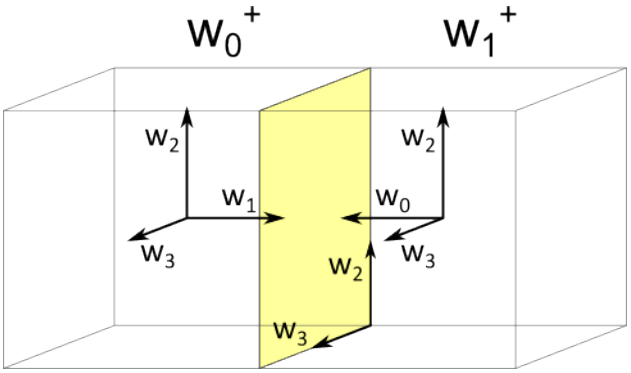


Figure 3.9: Relative orientation between the w_0^+ and w_1^+ C-cubes.

all planes intersect the Universe, as in row c). At left, the parametric point is displayed in red. The point lies in the border of the parametric hypercube, which is unfolded into a net of eight cubes -the C-cubes- in the 3D space. The point lies in the C-cube highlighted in red. The positive singular point is drawn in white. The negative one, in black, is split in four C-cubes due to the unfolding. When the point is near a face, this face and the equivalent face of the connected C-cube are both colored in yellow as it is shown in row b).

The distribution of the C-cubes in the net and the orientation of each local coordinate system fulfill the following properties:

- In a w_i C-cube, the face where $w_{j\neq i} = +1$ is the common face with the w_j^+ C-cube and where the value is -1, the neighbor is the w_j^- C-cube.

- The specific orientation of the local system of coordinates in a C-cube is such that the common face of two adjacent C-cubes has the same orientation in both systems. To accomplish that, system coordinates of the w_0^+, w_1^-, w_2^+ and w_3^- C-cubes are right-handed while the remaining ones are left-handed.

The relative orientation of the w_0^+ and w_1^+ C-cubes is shown in Figure 3.9. In the w_0^+ C-cube the positive w_1 axis points to the w_1^+ C-cube, and vice versa. The common face, in yellow, shares the same axis (w_2 and w_3) and the same orientation in both C-cubes.

The connection between C-cubes is also shown in Video 6 in `http://www.cs.upc.edu/~jsurinach/Thesis/Videos.html`. The parametric point is translated along an arbitrary but straight path and the corresponding plane is drawn in the right window. When the point approaches to a face -a common

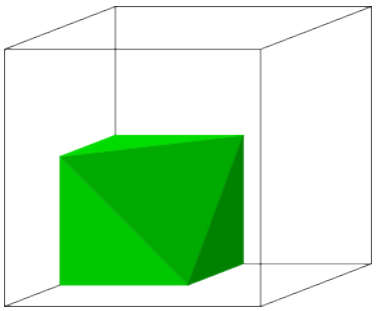


Figure 3.10: The green volume represents the parametric region in a C-cube where the plane does not intersect the Universe.

face of two connected C-cubes-, it turns to yellow (both faces if cubes are not adjacent in the net). The C-cube where the point lies on is highlighted in red.

Any plane π that intersects the Universe has at least one vertex in its positive half-space and at least another vertex in the negative one, therefore it must fulfill the boolean expression:

$$\Big(\bigvee_{i=0}^{7} \pi(V_i) < 0\Big) \ \wedge \ \Big(\bigvee_{i=0}^{7} \pi(V_i) > 0\Big)$$

being V_i the vertices of the Universe Cube.

The planes that do not intersect the Universe can be obtained by negating the previous expression:

$$\Big(\bigwedge_{i=0}^{7} \pi(V_i) > 0\Big) \ \vee \ \Big(\bigwedge_{i=0}^{7} \pi(V_i) < 0\Big)$$

The *and* of the linear inequalities becomes the intersection of their half-spaces -which are convex sets of points-, resulting in another convex set. Hence, the overall expression defines the union (*or*) of two convex sets.

Following the property of linearity of the CCPP -stated in Section 3.2.4-, a convex set in the euclidean space is also a convex set in the parametric space. In consequence, the set of parametric points corresponding to planes that do not intersect the Universe is the union of two convex sets.

The $\pi(V_i) < 0$ convex set includes the negative singular vertex whereas the $\pi(V_i) > 0$ include the positive singular vertex. Since a C-cube contains just one singular vertex (the w_i^+ contains the positive one and w_i^- the negative one), it has only one of the two convex sets. Due to the symmetric behavior of the four coordinates, the convex set has the same shape in all C-cubes.

Figure 3.10 represents any of the eight C-cubes in the parametric space. The singular point is the occluded vertex. The green volume is the convex set of parametric points where the corresponding planes does not intersect the Universe. The volume of the region is 5/48 of the C-cube. Video 7 in `http://www.cs.upc.edu/~jsurinach/Thesis/Videos.html` illustrates the non-intersecting region around the positive singular point (which is occluded). When a moving parametric point enters the region, the corresponding plane ceases to intersect the Universe, to return to intersect it when the point exits the region.

Note that the union of the four 3D convex sets of the C-cubes with the same singular point is not a 4D convex set. Each of the four 3D convex sets lies in an space orthogonal to the other three. Therefore, a straight line from a point in a set to a point in another set -supposing that these points do not lie in the common significant face- crosses the 4D space passing through points that do not belong to any of the 3D convex sets.

3.2.4 Properties of the Connected-Cubes parameterization of planes

Consider the base tetrahedron defined by four vertices of the Universe. Without loss of generality, let us assume that they are $P_0 = (0,0,0), P_1 = (1,1,0), P_2 = (0,1,1), P_3 = (1,0,1)$. The barycentric coordinates $(\alpha_0..\alpha_3)$ of a generic point (x,y,z) in the Universe are,

$$\alpha_0 = 1 - \frac{1}{2}x - \frac{1}{2}y - \frac{1}{2}z$$
$$\alpha_1 = \frac{1}{2}x + \frac{1}{2}y - \frac{1}{2}z$$
$$\alpha_2 = -\frac{1}{2}x + \frac{1}{2}y + \frac{1}{2}z$$
$$\alpha_3 = \frac{1}{2}x - \frac{1}{2}y + \frac{1}{2}z$$

Then, the scalar field value at (x,y,z) is,

$$\alpha_0 w_0 + \alpha_0 w_1 + \alpha_1 w_2 + \alpha_3 w_3 = ax + by + cz + d$$

Where the coefficients of the oriented plane $ax + by + cz + d = 0$ defined by $(w_0, ..., w_3)$ are:

$$a = -w_0 + w_1 - w_2 + w_3$$
$$b = -w_0 + w_1 + w_2 - w_3$$
$$c = -w_0 - w_1 + w_2 + w_3$$
$$d = 2w_0$$

Property 1: The Connected-Cubes Parameterization is Linear

As we can observe, the constraint that a point (x, y, z) is inside (or outside) the half-space defined by a plane is represented by a linear constraint $ax + by + cz + d < 0$ (or $ax + by + cz + d > 0$) in the parameterization space. This constraint is a linear half-space in some of the Cubes of the parameterization:

$$w_0^+ : \alpha_0 + w_1\alpha_1 + w_2\alpha_2 + w_3\alpha_3 > 0$$

$$\dots$$

$$w_3^- : w_0\alpha_0 + w_1\alpha_1 + w_2\alpha_2 - \alpha_3 > 0$$

Property 2: The Connected-Cubes Parameterization is faithful

A plane intersects a cube by 3 to 6 edges. Since a plane is defined by only three points, we use the intersection points of the plane with the edges and three of them are chosen to represent the plane. The edges are discretized in a number of voxels so the intersection points are approximated to the nearest voxel.

To ensure faithfulness, the discretization of the chosen edges must guarantee that any plane can also stabs the remaining intersected edges in all voxels of their discretization.

We work case by case depending on the number of intersections. Figure 3.11 shows the cases, excluding rotations and symmetries. Red lines are the intersected edges: thick edges are the three chosen ones to represent the plane and the faint edges are the discarded ones.

- **3 edges.** This case produce three points. There are no options and all three points are selected.

- **4 edges.** Four points are produced; one of them has to be discarded. There are two possible configurations of intersections: 4.1 and 4.2. In each configuration, all edges are symmetric and we can discard any one of them.

- **5 edges.** Even though there is only one configuration of intersections, two different options of chosen edges (5.1 or 5.2) are presented since only one of them alone cannot guarantee that all voxels in the remaining intersected edges (the red faint ones) are accessible.

- **6 edges.** In this configuration, edges form a continuous sequence around the cube. Alternated edges are chosen; two indistinct options take place.

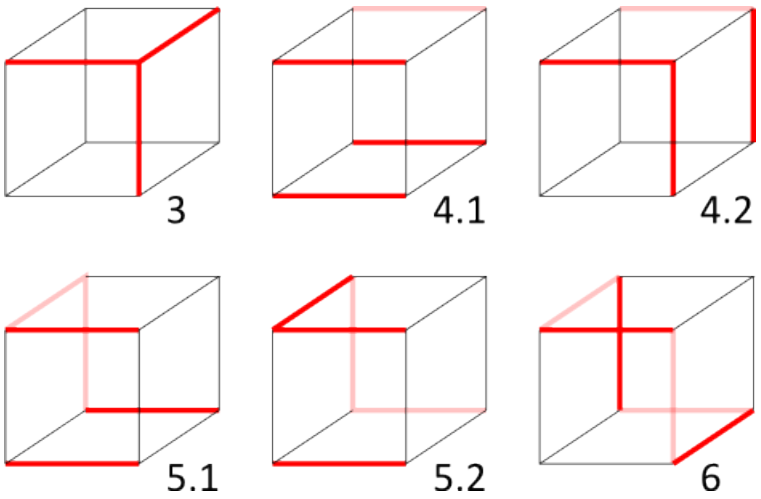


Figure 3.11: The possible intersections with a plane, and the sides of the Universe used to define faithfulness.

The faithfulness property guarantees a sufficiently dense representation of the planes in the Universe, by ensuring that the optimization algorithms do not lose valid solutions. More formally, we will say that a discrete parameterization of planes P is faithful if given any three voxels on any of the possible combinations of three Universe edges shown in thick red in Figure 3.11, there is a plane representable in the discrete parameterization that intersects these three voxels and also stabs the light red edges.

In order to detect the discretization in the Connected-Cubes parameterization that ensures faithfulness, an exhaustive test has been performed. The set of stabbed voxels on each of the red edges of the corresponding configurations in Figure 3.11 has been computed for each discrete sample in different discretizations ($N \times N \times N$, $1.5N \times 1.5N \times 1.5$, $2N \times 2N \times 2N$ and $3N \times 3N \times 3N$) of the cubes of the plane parameterizations.

The conclusion is that a discretization of $2N \times 2N \times 2N$ in the eight parameterization cubes is sufficient in order to ensure that the convex polyhedral regions contain at least one represented plane. This guarantees the faithful property.

Notice that this analysis can be reworded without considering the cases in Figure 3.11. The choice of representative edges is really only relevant to counting triplets without excessive redundancy.

Obviously, faithfulness with a $2N \times 2N \times 2N$ discretization of the Connected Cubes space as guaranteed by Property 2 is only obtained with the base tetra-

hedron shown in Figure 3.6. Any other base tetrahedron obtained by an scaling of the configuration in Figure 3.6 will be faithful (with the same $2N \times 2N \times 2N$ discretization) on a scaled cubic Universe representing the bounding box of the tetrahedron vertices P_0, P_1, P_2, P_3. Let us now assume that we have a fixed Universe with the canonical base tetrahedron in Figure 3.6, and that we scale this base tetrahedron by a factor $s > 1$ by computing $P_k^{'} = s \cdot P_k$ for $k = 0..3$. The new (non canonical) base tetrahedron $P_0^{'}, P_1^{'}, P_2^{'}, P_3^{'}$ will guarantee (with a $2N \times 2N \times 2N$ discretization) the faithfulness in a new expanded Universe that can be computed as the bounding box of the vertices $P_k^{'}$. However, and because of the Universe amplification, faithfulness on the initial Universe will be lost, due to the fact that for any possible voxelization, the size of the voxels of the initial Universe will be smaller than the size of the corresponding voxels in the expanded Universe. But, if we now scale the canonical base tetrahedron by a factor $s < 1$ by computing $P_k^{'} = s \cdot P_k$ for $k = 0..3$, the new (non canonical) base tetrahedron $P_0^{'}, P_1^{'}, P_2^{'}, P_3^{'}$ will also not be able to guarantee the faithfulness of the new contracted Universe. This is due to the fact that planes outside the Universe in Figure 3.6 are always represented with smaller precision. The conclusion is that any other base tetrahedron obtained by an scaling of the configuration in Figure 3.6 will not be faithful for the Universe in this Figure.

As a consequence, we will use the Universe and canonical base tetrahedron in Figure 3.6 in what follows.

Property 3: The Connected-Cubes Parameterization is 2-concise

A faithful parameterization of planes P is said to be c-Concise for $c \in \mathbf{N}$ if the total number of planes representable in that parameterization divided by the total number of triplets of voxels used to test faithfulness is smaller or equal to c.

A plane representation consisting of an integer identifying a choice of three edges of the Universe cube, the stabbing points on these edges and a bit for the orientation of the plane is (obviously) faithful but it is not linear. Also the naive parameterization of planes by their axis intercepts is non linear: if we describe a plane by the triplet of numbers (x, y, z) such that in a fixed reference system the points $(x, 0, 0), (0, y, 0)$ and $(0, 0, z)$ lie on the plane, then the condition that a certain point $P = (x_p, y_p, z_p)$ is on the plane is $x_p yz + y_p xz + z_p xy - x_p y_p z_p = 0$, which is obviously non linear in the parameters x, y and z (although it is linear on each of them separately).

From the property 2, we know that a discretization of $2N \times 2N \times 2N$ in the eight parameterization cubes is necessary in order to ensure the faithful property. This gives a total of $8(2N)^3 = 8 \times 8(N^3)$ planes that can be represented.

In contrast, the well-known Hough representation (spherical coordinates), must be discretized in a finer way to be faithful: the required discretization is $3N \times 3N \times 3N$. The Hough parameterization is 3-concise, with a total of $8(3N)^3 = 8 \times 27(N^3)$ represented planes.

According to Figure 3.11, any plane may intersect the universe cube in 3, 4, 5 or 6 edges, and the intersection cases can be reduced to only four configurations, as the cases 4.1 and 5.1 (resp. 4.2 and 5.2) can be represented with the same sides of the cube. Let us count the number of planes represented by each of these configurations:

- Configuration 3 represents N^3 planes (combinations of intersecting voxels in the edges), and this configuration appears in eight possible orientations in the Universe, around each of its corners. The total number of represented planes is therefore $8 \times N^3$.

- Configurations 4.1 and 5.1 represent N^3 planes each, and this configuration appears in twelve possible orientations in the Universe: Three possible orthogonal orientations, and four possible locations of the non-red edge in each case. The total number of represented planes is therefore $12 \times N^3$.

- Configurations 4.2 and 5.2 represent N^3 planes (combinations of intersecting voxels in the edges), and this configuration appears in twenty-four possible orientations in the Universe: Twelve edges, two configurations for each of them (only rotations must be taken into account, as symmetries switch to a different configuration). The total number of represented planes is therefore $24 \times N^3$.

- Configuration 6 represents N^3 planes, and this configuration appears in eight possible orientations in the Universe: Four pairs of diagonally-opposed vertices, two sets of alternating edges in the unique path that does not contain these two vertices in each case. The total of represented planes is therefore $8 \times N^3$.

In short, there are $(8 + 12 + 24 + 8) \times (N^3) = 52(N^3)$ planes represented by the configurations in Figure 3.11. As we are in fact representing $8(3N)^3$ planes, the Connected-Cubes plane representation is $8 \times 27/52$ concise.

Property 4: Loci of Parallel Planes

The planes with a given normal vector $\mathbf{n} = (a, b, c)$ are the planes fulfilling the three following equations, where λ is an arbitrary scaling factor:

$$\lambda_a = -w_0 + w_1 - w_2 + w_3$$

$$\lambda_b = -w_0 + w_1 + w_2 - w_3$$

$$\lambda_c = -w_0 - w_1 + w_2 + w_3$$

The loci of the set of parallel planes with this normal vector $\mathbf{n} = (a, b, c)$ is thus a linear 2D manifold in the 4D parameterization space, that corresponds to a line segment in some of the cubes of the parameterization.

In fact, given a normal vector $\mathbf{n} = (a, b, c)$, all parallel planes with $\mathbf{n}$ are represented by two straight segments in two of the cubes of the parameterization.

One of them belongs to a w_k^- Cube, and spans from its negative singular vertex to one of its significant faces (w_k^-, w_l^+). The second one belongs to the Cube w_l^+ and spans from the joining point in the face $w_0 = \cdots = w_3 = 1$.

The Gauss sphere gets divided into 12 disjoint parts, that univocally correspond to each of the 12 significant faces (w_k^-, w_l^+) of the Connected-Cubes Manifold. The set of significant faces of the Connected-Cubes form a closed, non-intersecting surface that is homeomorph to the Gauss sphere, Figure 3.12. This Figure represents two of the cubes of the parameterization and their common significant face (w_3^-, w_0^+), which maps onto the rombic region shown in the Gauss sphere. The set of parallel planes having their one of the normals in this region are represented by the two yellow segments spanding from the two singular points to a common point in the significant face.

3.2.5 Discretized Connected-Cubes Plane Parameterization

In our present implementation to maximize encoding accuracy, we locally relate the plane parameterization to cube C of the octree node, by using the vertices of the surfels' octree node to define a local frame and the vertices of the supporting base tetrahedron. The size and location of cube C is automatically obtained from the octree traversal information during the rendering process.

As a first option, we discretized the cubes of the boundaryless 3D manifold in 1024^3 voxels representing planes, so that oriented surfel planes can be encoded in 33 bits. We use 3 bits to encode the C-cube $w_0^+ \cdots w_3^-$ and $3 \times 10 = 30$ bits to encode the three coordinates of the oriented surfel plane in its C-cube.

Another option has been also tested. In this case the cubes are discretized in 256^3 voxels representing planes, involving an encoding of 24 bits, plus the

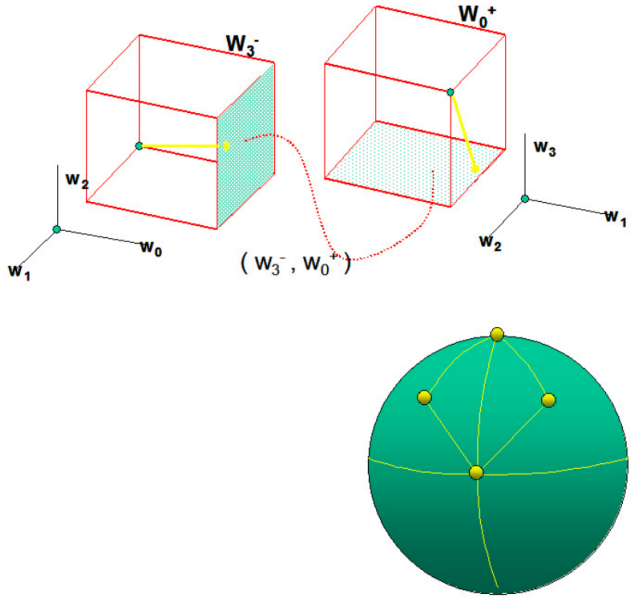


Figure 3.12: Two of the eight cubes and their joint significant face. This significant face (w_3^-, w_0^+) corresponds to the rombic region in the Gauss sphere represented below.

3 bits of the C-cube. In this situation the plane is encoded with $24 + 3 = 27$ bits.

Shading information also is stored in the surfel. The size of this data is 5 bits, as explained in Section 3.4. Therefore, the first encoding option (1024^3 voxels) leads to 5 bytes per surfel and with the second one (256^3 voxels) only 4 bytes are required. A detailed discussion of encoding options is presented in Section 5.4.2.

3.3 Color of Surfels

In a Medical Atlas the fidelity to reality is of main importance in all its details, being color one of them. The color of the rendered model is one of the most significant elements to confer realism to the image.

There are some intrinsic problems in the precise color acquisition for medical models, as it is pointed out by ap Cenydd et al. [186]. Among them: images obtained in vivo from organs are irrigated by blood, giving them a reddish color -while its absence in cadaveric dissections leads to grey tones-; the continuous irrigation of the area with a saline solution in surgical procedures produces a large uniform specular reflection. Also, organ's features -texture, surface

normals, translucency...-, and the environment -direction from the camera, lights- are additional elements to be taken into account.

In our model, the Visible Human dataset, the color of sampled points is explicitly stored in the model as jpeg images. This information has been extracted and encoded as RGB values -8 bits per channel- along the segmentation. Therefore, for any sampled point, its segment value and color are obtained at the same time.

Four different shading models of surfels are analyzed in this work. All of them are based in the *basic color* of the Surfel. The method used to obtain this basic color has two phases: the color of the high resolution Surfel comes from the color of the sticks, which are in turn computed from the color of the samples.

A stick is the line that joins two adjacent samples -the endpoints-, with a color in each of them. The endpoints are classified as *in* or *out* depending on the considered organ. The color of the *in* endpoint is assigned to the stick. We note that the *in* endpoint usually does not lie on the organ boundary but just below it -in the direction to the interior of the organ-, so this is not the actual color of the surface. However, assigning the color of the *out* endpoint or weight-averaging both colors can produce strange and inaccurate colors in the organ boundary.

Finally, the color of all sticks in a voxel are averaged to produce the basic color of its Stick. The averaging is computed independently for every channel.

The color is transformed and processed in the YCoCg-R color space (Luminance, Orange chroma, Green chroma), that decouples luminance -intensity- and chrominance -color information-. A model with separated luminance is preferred because this channel is used in one color model of the surfels. The YCoCg-R space is preferable to YCbCr for a practical reason: computations use only integer arithmetic -additions and shifts-, much simpler than floating-point calculations of the YCbCr model. Moreover, the YCoCg-R space is lossless reversible from/to RBG, in contrast to YCoCg.

Constrained Surfels will be used in the framework of a hierarchical subdivision model, as discussed in Chapter 4. The next four subsections discuss different encoding solutions for coarser surfels in upper regions of the multiresolution tree.

3.3.1 Uniform Color

In this model all Surfels are monochrome. They present no chromatic difference in their surface. The color of a high-resolution Surfel is the *basic color* described previously. In a surfel of an inner (upper) tree node, its color is averaged from the color of the surfels of its children.

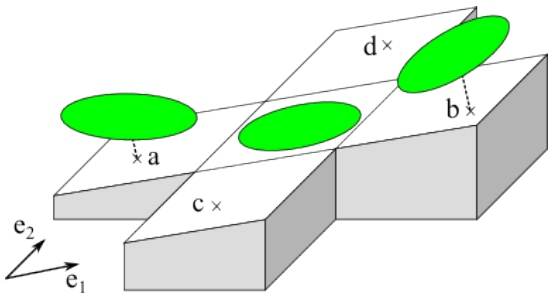


Figure 3.13: Central node and four neighbors. Nodes are clipped by the plane of the central node. Points a to d are the projections of central points of neighbor surfels to the plane. Surfels c and d are not shown for clarity.

This is a simple model, used for comparison purposes only.

3.3.2 Gradient of Intensity

Uniform surfel colors result in an unpleasant piecewise flat appearance. Instead, in this approach the color of each fragment in the surfel is interpolated through a precomputed color gradient which is stored as part of the surfel information. The color gradient is computed from the color of four neighboring nodes. Given the surfel's normal vector $\vec{n} = (n_x, n_y, n_z)$ and being $|n_\alpha|$ its dominant component, $|n_\alpha| \geq |n_\beta| \geq |n_\gamma|$ where (α, β, γ) denote a permutation of (x, y, z), neighbors are looked up in both directions of axes β and γ. If any of those neighbors has no sticks (due to the bending of the surface), then four more candidates for that neighbor are sequentially sought: an offset of $\pm$ times the side of the node in the direction of axis α is applied to the void neighbor, and new candidate nodes are obtained. Finally, duplicates are rejected. The central points of the neighbors are then projected onto the plane of surfel of the considered node (Figure 3.13). A linear gradient of the luminance component of the colors is computed from these points and the color of their surfel center by solving a linear system of two equations, as the gradient is a 2D vector which lies on the plane of the surfel. Our 2D frame for gradient representation consists of two orthogonal vectors, the first one e_1 being the projection of the world coordinate direction γ on the surfel plane. Obviously, the second intrinsic direction of the plane frame is the cross-product between $\vec{n}$ and e_1. By using this implicit frame we are able to decode and use color gradients in the client without sending any information on the surfel plane frame.

We use the color of the surfel centerpoint and the intensity gradient to compute colors at all points of the surfel disk. These points inherit the compo-

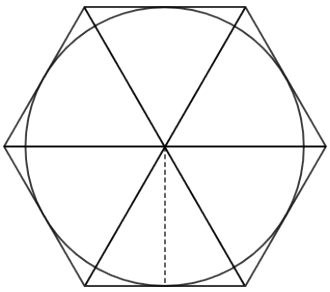


Figure 3.14: Hexagon circumscribed to the Surfel.

nents Co and Cg of the centerpoint, while the intensity Y is modulated by its gradient and location. Given a surfel s, the fragment which renders the center point $s.P$ inherits the surfel basic color $s.c$ whereas the color $f.c$ of any other fragment f is computed from $s.c$ and the gradient $s.g$, depending on the vector from $s.P$ to the fragment position $f.P$ within the plane $s.pl$:

$$f.lum = s.lum + s.g * (f.P - s.P)$$

where $f.lum$ and $s.lum$ are Y luminances in YCoCg-R color space.

We have observed that using Co and Cg gradients does not improve the visual quality.

3.3.3 Color per vertex

This method is based on the interpolation of points on the surface of a convex polygon.

The Surfel is decomposed in adjacent triangles arranged as a fan sharing a central vertex. The resulting figure is a polygon, its center being the center of the Surfel. In this option, the figure chosen is a regular hexagon, so triangles are equilateral and their height -the apothem of the hexagon- is the radius of the Surfel. Hence, the Surfel is inscribed in the hexagon. See Figure 3.14.

The colors of the six vertices are obtained by projection: the image from the uniform color model at the highest resolution (Section 3.3.1) is rendered and every vertex of the hexagon is projected on the image. Then, the color of the image in the projected point is retrieved from the framebuffer and assigned to the vertex. The Surfel encodes the color -in the YCoCg-R space- of the six vertices plus the color of its center, which is the basic color of the surfel.

The pipeline receives the six triangles and renders all their fragments using bilineal interpolation. Since adjacent triangles share two vertices -and the edge between them- no color discontinuities appear among triangles.

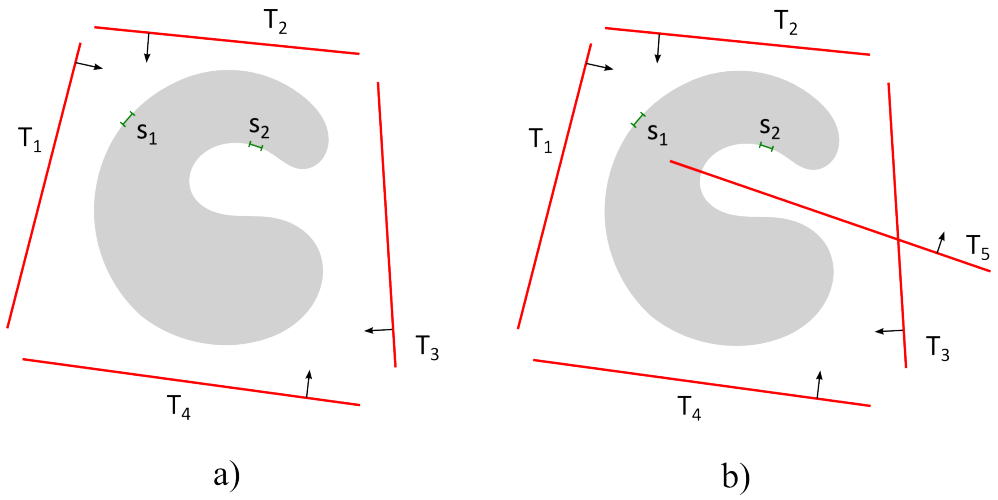


Figure 3.15: a) Surfel s_1 is captured by both T_1 and T_2 but is projected to T_1. s_2 is occluded. A rendering with false color is performed to detect such surfels. b) New texture image T_5 is generated to capture s_2.

The fragment shader is responsible for rendering only the fragments whose distance from the center is less than the radius, giving a circular appearance to the surfel.

3.3.4 Projective Texture Mapping

In this case (*PTM* in the following), we compute a set of texture images T_1.. T_n from the high resolution surfels of the segmented volume V by rendering all maximum resolution surfels from n initial directions, see Figure 3.15. The rendering direction corresponding to any T_i is always the normal direction of the T_i plane. Some surfels such as s_1 will be captured by more than one image (T_1 and T_2 in this case), but some other surfels like s_2 in Figure 3.15-(a) will not be captured due to occlusions. We detect these situations by performing a second render of the surfels with false color, and we add a few more directions and texture images T_k in a way that surfels like s_2 become visible, Figure 3.15-(b). By computing surfel importance in every image T_k as dot products between surfel normals and render directions, we are able to compute the best texture image for every surfel. At the end of the preprocess, surfels encode their plane equation and a one-Byte index k to their texture T_k. During render, surfel appearances are computed by projecting T_k onto them in the texture normal direction. This is valid for surfels at any level in the multiresolution tree.

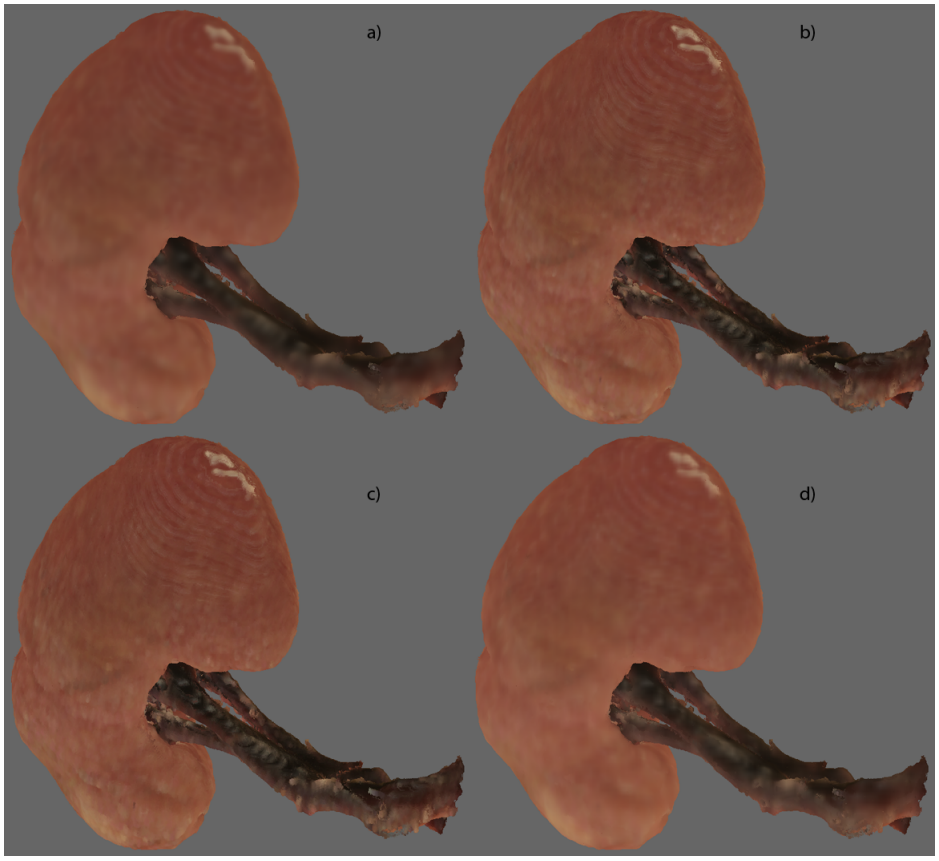


Figure 3.16: Visual comparison of the four color methods. a) Uniform color. b) Color per vertex. c) Gradient of Intensity. d) Texture mapping.

We initially test twenty textures for any organ. The projection points of view are the center of the triangles of a regular icosahedron circumscribing the organ, aiming at the center. The icosahedron guarantees regular and symmetric points of view of the images.

3.4 Comparative Analysis: Color of Surfels

The four methods previously detailed are alternative surfel rendering schemes. An analysis of the methods has been done and results are summarized in this Section. The four methods have been compared using a number of examples, being Figure 3.16 -a kidney and the renal vein rendered with high-resolution surfels-, one of them.

Color method	**Stored elements**	**Memory consumption**
Uniform	1 color(1)(2)	8+9+9 = 26 bits = 3.25 bytes
Gradient	1 color + 2D gradient	26+2×32=90 bits = 11.25 bytes(3) 26+2×16=58 bits = 7.25 bytes(4)
Color per Vertex	1 + 6 colors(5)	26×7 = 182 bits = 22.75 bytes
PTM	1 index(6)	5 bits

Table 3.1: Memory consumption per surfel of shading information in the four color methods.
(1) The color of the surfel. All methods, except PTM, contain this field.
(2) Channels: Y - 8 bits, Co - 9 bits, Cg - 9 bits
(3) Assuming 32 bits float numbers.
(4) Float numbers quantized to 16 bits.
(5) Center plus 6 vertices
(6) Discussed in Section 4.4.1

Visual appearance

The *Uniform* color method is obviously the method that produces the worst results. Even though high resolution surfels are small enough to show an acceptable appearance when the point of view is far from the organ, when the user takes a closer look of a detail or surfels become progressively bigger, the piecewise flat aspect of the surface is the overall perception.

The results of *Gradient of Intensity* and *Color per Vertex* are pretty similar. Non monochromatic surfels reduce the piecewise effect but the overlapping areas of adjacent surfels are not as smooth as it is expected. Especially in regions of the model with high curvature radius where the overlapping is clearly visible.

The results of *PTM* are the best of all methods. The overlapping of adjacent surfels does not introduce artifacts since it is very likely that both surfels map to the same texture. Otherwise, the textures are close enough to not produce visible differences between surfels. In addition to that, since the same texture is used by surfels of all levels, the appearance is very similar even at coarser levels of the octree.

Memory consumption

Results of shading information per surfel are shown in Table 3.1. Note that values are not similar and they differ by a factor that ranges from two to three.

Two results are shown in the *Gradient* method. In the first one the intensity gradient -two values, since it is a 2D gradient- is implemented with 4-byte floats. We have made a second test quantizing floats with 2 bytes, obtaining very similar results. Therefore, the second result is also presented.

In the *PTM* method, the storage of the textures is omitted here but it must be taken into account to obtain the overall occupation. The 5 bit result accounts for interactive information transmitted per surfel, since textures are sent only once -to a client device- at the beginning of the session, as we state in Chapter 5.

Judging from the results presented in the table, the most convenient method is *PTM*. *Uniform*, in second place, is discarded due to its bad visual results. *Gradient* and *Per Vertex* are the worst performers.

Overall result

Given that Projective Texture Mapping is the color scheme that surpasses the other ones both in visual appearance and in memory occupation, it is clear that this method is the best choice and it will be used in the rest of our work.

3.5 Conclusions

In this Chapter, a special type of surfels -*Constrained Surfels*- has been presented. Part of the geometry of these surfels is inferred from a spatial subdivision model. This results in a lesser memory occupation of surfels. Constrained Surfels can be completely determined from their supporting plane and the geometry of their corresponding cells in the space subdivision.

We introduce a parameterization of the supporting plane of these surfels using the Connected-Cubes Plane Parameterization technique. This method has a number of relevant properties, leading to an efficient and compact representation of surfels as we will see in next chapter.

Four different color options of surfels have been analyzed in detail. A comparative study of these options has been performed and Projective Texture Mapping is the method that produces the best color approximation and is the option used in this work.

Chapter 4

Surfel Octrees

In the present Chapter, we discuss specific data structures for the interactive inspection of segmented medical models, with the goals of having a multiresolution visualization of the organs and low memory footprint.

Given that organs do not intersect, there are two options to structure a scene with multiple organs: one octree that includes all the organs, or one octree per organ and a forest of octrees that encompasses the complete scene. Our approach builds an octree for each organ because separated octrees give flexibility and make the independent management of organs possible.

In this Chapter, we present the construction of Constrained Surfels from binary segmented volume models and their subsequent simplification aimed to populate the Surfel Octree. These actions take place in a preprocess phase, in which the Octree Forest is computed. As already mentioned, the Octree Forest is a set of organ octrees. We process one organ at a time in order to compute its corresponding octree.

A compact representation of Surfels and the Surfel Octree is also detailed. This is a representation that achieves an efficient transmission rate and improves the response of client devices.

4.1 Definition

Surfel Octrees are a compact representation of segmented volumes. They recursively subdivide space into Void and Grey nodes and store this subdivision in a compact way. Void nodes correspond to cubic regions of space that are not intersecting the boundary surface of the organ being represented while Grey nodes are cubic regions that do intersect this boundary surface. By approximating the organ boundary surface within each Grey node by a planar surfel, Surfel Octrees become multiresolution compact representations of the

corresponding organs. Surfel Octrees are generated in two steps. In the first one, the organ bounding cubic box is recursively subdivided and the octree structure with Void and Grey nodes is created in a top-down way. Subdivision stops when the tree node size equals the distance among samples. Then, in a second step, surfels in the leaf Grey octree nodes are computed by estimating the smoothing normal vectors in a similar way to [193] and non-leave Grey octree nodes are computed by an efficient bottom-up simplification process, also based on the Organ Octree generation algorithm, [193].

4.2 Generation

The data used to construct Surfel Octree Atlases is a segmented volume model without density information. The samples are arranged in an axis-aligned homogeneous grid. For a specific organ, the segmentation produces a binary volumetric model where a sample is classified as pertaining or not pertaining to that organ. They are labeled as *in* and *out*, respectively. The collection of *in* samples provides an approximation of the organ.

We define a *voxel* as the axis-aligned cube with adjacent samples in its vertices.

A Surfel Octree is created for each organ. The octree consists of a hierarchical partition of the space occupied by the samples of the organ, the voxels of the model being its leaves.

The model is processed with the aim to create a Surfel in every voxel; this surfel is assigned to the corresponding (leaf) node. An ulterior process creates surfels in interior (non-leaf) nodes.

4.2.1 Sticks

We consider the segments that connect any sample to its six axis-aligned neighbors. Having being extracted from a homogeneous grid, all segments have the same length, which depends on the sampling frequency. A segment has two endpoints, each one corresponding to a sample. Segments that have an *in* endpoint and an *out* endpoint are defined as *sticks*. Sticks lie at the edges of voxels.

The boundary surface of organs is continuous. Therefore, this boundary stabs all sticks. Inversely, the surface intersects the stick at a crossing point belonging to the segment. The error is limited by the length of the segment. A stick defines an interval for the local geometric surface boundary, as exact intersection points are unknown.

Sticks are processed to approximate the local geometry and orientation of the surface. We use them to calculate high-resolution Surfels, which populate the nodes of the lowest level of the octree.

4.2.2 Filtering and Normal Estimation

Sticks become the geometrical limits where the boundary of an organ lies. Although sticks have a small size, this alone is not accurate enough to define high-resolution Surfels. This situation derives from the binary segmentation, where no scalar field provides geometric information between samples. Two methods are proposed to produce realistic representations and to avoid staircase artifacts.

Filtering

To produce a better and more realistic surface, we apply a constrained smoothing method: crossing points of the boundary are modified in order to increase the surface fairness, while these points are constrained to remain in the stick's interval. With this constraint, the faithfulness of the boundary to the geometric information of the model is guaranteed.

General smoothing algorithms have been described in Section 2.2.4. We recall that Gibson [48] constrained the vertices of the smoothed isosurface to be placed inside their voxel. After a nonlinear optimization algorithm that minimized an energy function, Nielson et al. [192], restricted resulting vertices to be placed also on sticks. In a later work Nielson [54] introduced a smoothing operator based on the dual of the dual surface of a Marching Cubes mesh, locating the vertices at the intersections of the dual's quads with the lattice edges.

We have chosen the constrained bilaplacian smoothing method from Chica et al. [68] because of the properties of the bilaplacian operator. It is also based on sticks extracted from binary voxelizations. The crossing points are initially assumed to be located at each stick middle point. A stick lies in two perpendicular axis-aligned planes (see Figure 4.1). Isocurves that connect crossing points of consecutive sticks are computed in each plane. A Laplacian smoothing operator is applied twice in each plane and the crossing point is displaced in that plane. The two resulting points are combined and projected to the line of the stick. If the point lies outside the interval of the stick, it is clamped to the nearest endpoint. This bilaplacian step is repeated until convergence is observed.

The initial position of crossing points -in the middle of the stick- produces

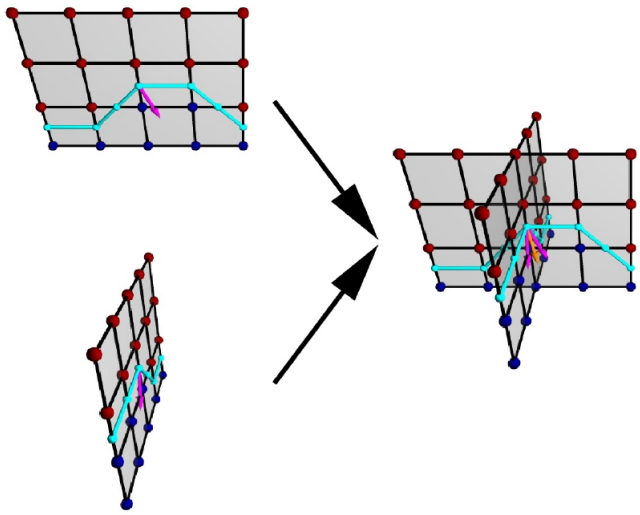


Figure 4.1: Bilaplacian smoothing operator. A Laplacian operator is computed in two axis-aligned orthogonal planes. Both results are combined and projected to the stick. Figure from [68].

a staircase effect and sharp edges. The bilaplacian operator adjusts the points to smooth edges. Figure 4.2 exemplifies the smoothing effect in edges: black lines are sticks, black and white circles are in and out endpoints, respectively. In blue, the isoline and crossing points. The dashed line shows the new isoline after the Laplacian operator. In red, the displacement of the crossing point. The angle formed by the isoline in the p crossing point has been flattened.

The effect of the bilaplacian method is an overall smoothing of the boundary while geometric restrictions are maintained. Since the organs are smooth, this method is well suited for our purpose.

In a second step, the estimation of the surfel's normal requires the previous determination of the normal of the boundary at the crossing point in each stick of the voxel.

Normal estimation

Once the crossing point p of the boundary and the stick is approximated, an estimation of its normal is computed. We define the *voxel ring* of a stick as the four adjacent voxels that share the same stick. For each of these voxels, a triangulating surface is computed using their sticks and crossing points and connecting them to p. Normals are recovered from each triangle. Finally, they are averaged and normalized. The resulting normal is associated with that

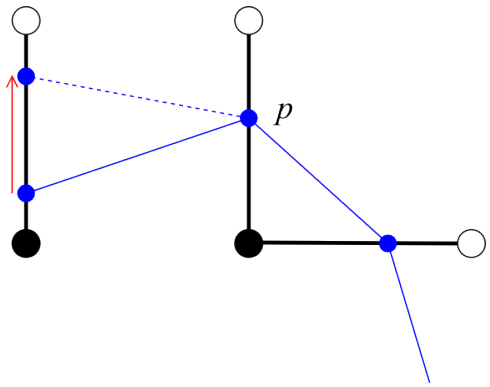


Figure 4.2: Laplacian smoothing operator. The red arrow shows the displacement of the crossing point due to the Laplacian operator. The new isoline shows a softer shape.

stick.

Later, a Laplacian filter -iterative weight averaging- is computed with the normal of the stick and the neighbor's averaged normals. This is the final normal of the stick.

This Laplacian filter is equivalent to incorporating the information of the crossing points of higher order voxel rings, that is the set of rings of all sticks that belong to lower-order rings of the stick.

The leaves of the octree share the same space as the voxels, so there is an equivalence between voxels and nodes. Surfels are associated to all voxels and their parameters are stored in the corresponding node. We next describe the definition of high-resolution surfels, which populate the leaves of the octree. Surfels of upper level nodes are described in the next section.

The normals of the sticks placed on the edges of the voxels are averaged and normalized again to obtain the normal vector of the surfel corresponding to the voxel. Note that every stick participates in up to four surfels, those of its ring of voxels.

Further details on the averaging calculation of surfel geometry are described in Section 4.3.1.

4.3 Surfel simplification

Surfels that populate upper levels of the octree, inner nodes, are generated from the lowest level Surfels using the simplification scheme described below.

The Surfel in a parent's node is the simplification of the surfels in its chil-

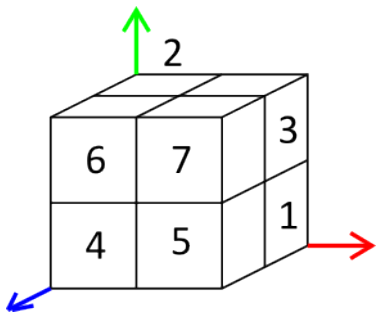


Figure 4.3: Order of recursive traversal of the nodes of the octree. Node 0 is below node 2 and behind node 4.

dren. Only Grey nodes -non Void ones- are considered.

A node is determined by its size and position. The position of the node is the geometric coordinates of the vertex closest to the origin of coordinates. This vertex is named the *origin* of the node.

A recursive traversal of the octree is defined by specifying an (arbitrary) order in which the eight children are visited. In this work, the order used is that shown in Figure 4.3. A *path* to a node is defined by the sequence of order numbers of nodes visited from the root to that node. In a recursive traversal, the path of every node is known at the time it is visited.

Therefore, with the knowledge of the origin of the octree and its size, and with the path to a node, the size and position of this node can be determined.

A Surfel is defined by its supporting plane, the position of the center and the radius. However, as already discussed in Chapter 3, surfels in Surfel Octrees can only be represented by its supporting plane.

Supporting plane

The calculation of the supporting plane of high-resolution surfels in leaves was explained in Section 4.2.2. In interior nodes, the normal vector of the plane and its displacement are computed separately. The normal is averaged from the normals of its children, see Section 4.3.1. The displacement of the plane is defined such that the defined volume in the node is the same as the sum of the volumes of their sons. Details are explained in Section 4.3.2. Finally, the parameterization of the plane is stored in the node.

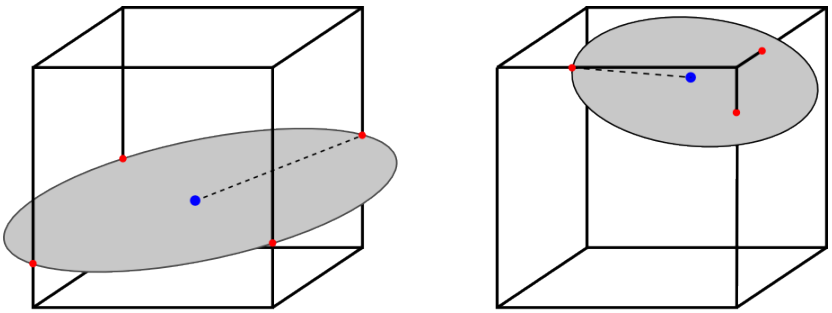


Figure 4.4: Constrained Surfel to its node. Red points are the intersection of the supporting plane and the edges of the node; the blue point is the barycenter and center of the surfel. The dashed line is the radius.

Center and radius

The position of the center of the Surfel and its radius can be obtained from the node information, thus there is no need to be stored in the node.

During the octree traversal, all nodes are visited and their path is known. The size and position of each node can be computed in a straightforward way, [96].

The center and radius of the surfel can be computed as follows: the cubic node is intersected with the supporting plane and intersection points with edges are determined. The barycenter of these points is defined to be the center of the surfel. Note that it also lies in the plane. The maximum distance from the center to the intersection points is the radius. In Figure 4.4 two examples are shown. In a) the distances from the center to intersection points are the same; in b) they are not. This radius produces a slight overlap with neighbor surfels and reduces the gaps among them.

4.3.1 Geometry

We use the fact that the Connected Cubes Plane Parameterization represents linear scalar fields to efficiently compute plane averages. In fact, given two planes π_1 and π_2 represented respectively by $(w_0^1..w_3^1)$ and $(w_0^2..w_3^2)$ where $w_k^i, k = 0..3$ are field values of the plane π_i at the four vertices of the reference tetrahedron, the plane represented by $(w_0^1 + w_0^2..w_3^1 + w_3^2)$ is obviously an average of π_1 and π_2. We use the invariance field property that says that a plane π_1 has infinite linear scalar field representations $(\lambda w_0^1..\lambda w_3^1)$ for any

$\lambda > 0$, as all of them have the same zero isosurface.

Consequently, we compute the supporting plane of high-resolution surfels by simply adding the weights of the CCPP encoding of the planes (crossing points plus normal vector) of the voxel sticks.

Similarly, we obtain the normal vector of the surfels of parent nodes during simplification by adding the weights of the CCPP encoding of the supporting planes of the surfels of the children nodes. This produces a first approximation π_p of the supporting plane of the parent surfel. The volume preservation constraint (see next Section) uses the property that the weights of any plane parallel to π_p can be replaced by the weights $(w_0^p + \alpha ... w_3^p + \alpha)$, as linear scalar fields with parallel isosurfaces can be represented by uniformly displaced weights. In short, once children plane encodings have been added and the first approximation weights $(w_0^p ... w_3^p)$ have been computed, the displacement α is computed to preserve the interior volume, and the CCPP encoding of the parent supporting plane is obtained.

4.3.2 Volume preservation

The calculation of the normal vector of the supporting plane has already been described. For the complete definition of the plane, its displacement remains to be determined or, alternatively, a point belonging to the plane. The rest of this Section presents the computation of this displacement in interior nodes.

Volume of a node

The boundary of the organ defines a space partition in Grey nodes: the semispace that is inside the organ -where the *in* samples are- and the space that is outside the organ. The *volume of the node* is the volume of the semispace inside the organ. We assume the simplification that the boundary of the organ is approximated by the supporting plane of the surfel in that node. By definition, a voxel -and a leaf node- have a side length of 1.

The volume of Void nodes (there is no boundary inside them) is easily determined:

- Node completely outside the organ: 0
- Node completely inside the organ: $side^3$

A simple verification of the segmentation value of any sample in the node shows the type of the node.

The volume in Grey nodes must be computed. We do it as follows.

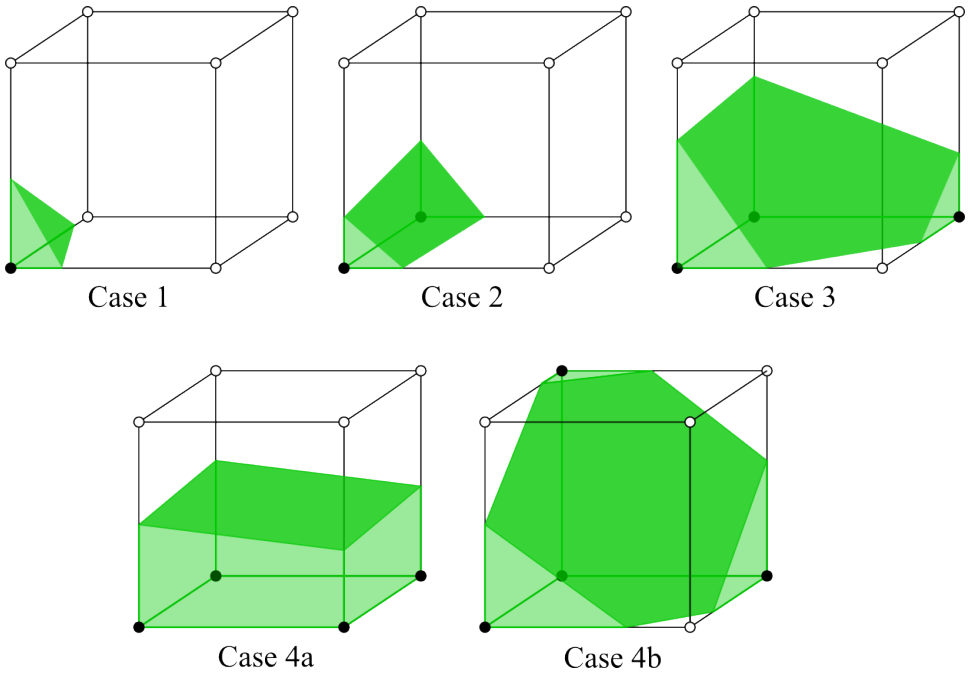


Figure 4.5: The 4+1 cases in volume calculation after symmetries and rotations. Black vertices are *in* points, white ones are *out*. In dark green, the plane. In light green, the volume.

The supporting plane intersects the edges of the node, determining a convex 3D object with planar faces -the inside part of the voxel-. In some situations, this object has a known formula to calculate its volume. Otherwise, we cut it in smaller objects. The final volume is the accumulation of those smaller volumes.

We work case by case depending on the number of *in* vertices. The direction of the normal vector of the plane defines which vertices are *in* and which ones are *out*. Figure 4.5 shows the cases.

- **Case 1**. 1 vertex *in*, 7 vertices *out*. The object is a tetrahedron with a right trihedral angle (three faces that form a right angle between any pair of them). Its volume is:

$$V_1 = \frac{abc}{6}$$

being a, b and c the three heights of the tetrahedron. In this case, the

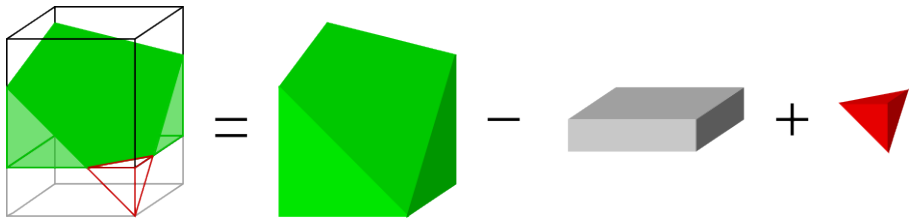


Figure 4.6: Case 3. The desired volume (at left, in green) is the decomposition of a sectioned prism minus the square prism base plus the tetrahedron.

heights are the length of the three concurrent edges to the vertex of the right trihedral angle (the *in* vertex).

- **Case 2**. 2 vertices *in*, 6 vertices *out*. The object is a truncated triangular pyramid. The bases are right-angled triangles. Its volume is:

$$V_2 = \frac{h}{12}\left(2a_0b_0 + 2a_1b_1 + a_0b_1 + a_1b_0\right)$$

being h the height of the pyramid -the side of the node-, and a_i and b_i the sides of the two triangles.

- **Case 3**. 3 vertices *in*, 3 vertices *out*. The object can be described as a sectioned prism minus the square prism at the basement plus a tetrahedron. Figure 4.6 illustrates this CSG-style decomposition. All these objects have a known volume, except for the prism section. Since the section crosses two opposite vertices, the volume of the prism can be easily computed.

- **Case 4**. 4 vertices *in*, 4 vertices *out*. Two different situations arise in this case:

  - **Case 4a**. The plane intersects 4 faces. The object is decomposed in a sectioned prism (seen in Case 3) plus the square prism at the basement.
  - **Case 4b**. The plane intersects 6 faces. The object is decomposed in a sectioned prism again minus two tetrahedrons, one at the top and the other at the bottom. The volume is computed as in Case 3 minus the top tetrahedron.

The cases of 5, 6 and 7 vertices *in* are symmetric to 3, 2 and 1 vertices *in*, respectively. The volume of their symmetric cases is computed and subtracted from the whole node volume.

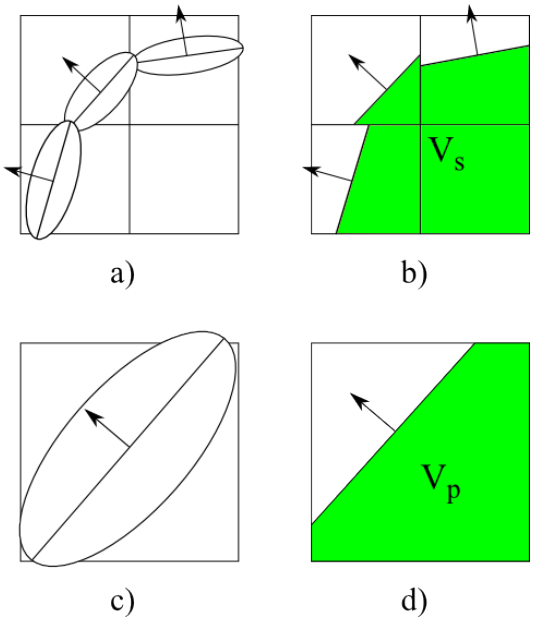


Figure 4.7: Volume preservation example in 2D. a) Three surfels defining the boundary of the organ. b) Supporting planes of surfels and the volume, in green. c) Surfel of parent node. d) The position of the plane is adjusted so the volume V_s of the green area in d) is the same as the volume V_p of the green area in b).

Preservation of the volume of children

We recall that the volume of a node depends not only on the orientation of the plane -the normal vector- but also on its position -the displacement along the normal direction-. Our objective is that the volume of a parent node must be equal to the accumulated volume of its children (see Figure 4.7). This is accomplished by adjusting the position in the parent node through the displacement α; the position of leaves is fixed and we do not modify it. The normal vector of the parent's plane has been already determined (Section 4.3).

For a given inner node, the volume of its children is computed and accumulated. Void nodes that are inside the organ are also taken into account. The resulting value is the volume that the parent node must preserve.

Finally, the plane displacement is computed in order to ensure that $V_p = V_s$, see Figure 4.7.

To work properly, this method is applied bottom-up in the octree.

4.4 Efficient compact representation

Data structures computed in the preprocess -the octree structure and the surfels- must be sent to the client in a portable device. It is of major importance that bandwidth occupation through the net is kept low, and the space required in the device to store the data does not exceed the available memory, which usually is not large.

The main problem with previous approaches like Organ Octrees [193] is twofold: individual surfels require an 11 Byte encoding, and the application must also take into account the storage overhead due to the octree structure. Our present proposal addresses these two problems in a novel and efficient way, as presented in the next paragraphs.

4.4.1 Constrained Surfels

The data required to determine a Surfel is its geometry and an associated texture. The amount of data required for each element is analyzed in the rest of this Section.

Geometry

Constrained Surfels are defined in association with a node, which contributes with its own geometry, obtained when traversing the octree. As stated in Section 4.3, only the supporting plane information (normal vector and position) has to be explicitly stored to recover the Surfel.

The Connected Cube Plane Parameterization provides an efficient method to encode planes. In Chapter 3 it was shown that a plane can be encoded by three plane coordinates and one integer in the range 0-7.

The integer is encoded with 3 bits and each of the three coordinates is quantized to N bits. A bigger value of N produce a more precise quantization, but at the cost of more memory consumption. Experiments discussing different values of N are presented in Section 5.4.2.

Texture

Every organ requires about 20 textures to be properly displayed. These 20 images are sent only once, at the beginning of the session. After that, there are no more transmissions of textures during the rest of the session. Therefore, when a surfel requires a texture to be rendered, that texture is already in client memory. The only required information is the index of the texture associated to the surfel.

Such information (1 value out of 20) can be encoded with 5 bits ($2^5 = 32$ possible values), which are greater than the 20 possible values needed. There is an excess of capacity that is used to codify extra textures when they are required.

Final representation

A Surfel is encoded with its geometry and texture information. Assuming that a plane is usually defined by 4 scalars and every one of them is codified with at least 4 bytes, and adding 5 bits for the texture index, that makes 133 bits ($\sim$17 bytes). Encoding schemes can obtain better compact representations; several proposals are presented and discussed in Chapter 5.

4.4.2 Octree structure

The Surfel Octree is a *Maximal Subdivision Tree*. Non-void nodes (nodes with relevant data) are subdivided until the deepest level is reached. In Surfel Octrees, this is the level where the space associated with the node coincides with a voxel.

Any tree contains two types of data structures: the tree structure and the node associated data. The former includes the information to retrieve the hierarchy and facilitate traversals (children, parent, level...) the latter is the repository of application-dependent data, which in our case consists of Constrained Surfels.

In this Section, the tree structure is analyzed.

Since Surfel Octrees are used in client-server applications, we focus on two objectives: the progressive transmission of data, and the efficient storage of the tree structure and its transmission over a network.

Progressive transmission of data

1. The tree structure and the data structure are kept separate. This is a relevant principle in the progressive transmission scheme because this scheme allows:
   (a) The transference of only selected surfels, not all of them.
   (b) The dumping of unneeded surfels, saving space in the client.
2. Transmission of the octree is done in two steps: the tree structure is transmitted only once, at the beginning of the session; and selected surfels (a LOD of the organ, for instance) are transmitted on demand during the interactive session.

Efficient storage and transmission

In Surfel Octrees, each node must provide its size and the geometric position of its *origin* (the vertex closest to the origin of coordinates). This information is deduced in a recursive way: the size of a node is half of its parent's size; the position depends on the parent's position plus an offset -the size of the child- that depends on the ordering of the children in the tree.

The size and position of a node are therefore deduced from the size and position of its parent. Given the recursive arrangement of the tree, this information must only be explicitly stored for the root node. That is, only once. The information of the remaining nodes is computed during octree traversal from path tracking.

One of the key ingredients of the Surfel Octree representation is the use of an extremely compact encoding for the octree structure, which has been inspired by [147].

We encode the octree structure as a single bit array -a bit stream-, with one Bit per node. A "0" is assigned to Void nodes and a "1" to Grey nodes. Octrees are encoded as a sequence of octree levels, each level k being a block of bits that represent all octree nodes at that level. The number of bits of the $(k+1)$-block is obviously $8 * g_k$ where g_k is the number of Grey (or "1") nodes in the previous k-block. Sets of 8 bits corresponding to the children of all Grey nodes in the k-block are encoded in the $(k+1)$-block in the same order their parents had on the k-block.

The bit stream is the sole data structure required to encode the tree structure of the Surfel Octree. An encoder process reads the model and creates the octree in the form of bit stream; a decoder process reads the bit stream to reconstruct the initial hierarchy.

Some properties of the hierarchy can be derived from the bit stream definition:

- Siblings stay in consecutive positions.
- The children of the i-th "1" of level k begin at the $8i$ bit of level $k+1$.
- The parent of the i-th bit in level k is the $(i \ div \ 8)$-th "1" in level $k-1$.

Figure 4.8 depicts an image (a) with its corresponding quadtree -2D counterpart of the octree-, (b). In (c) we show the bit stream of the quadtree. Bits are grouped and underlined per level to facilitate their comprehension. The bit of the root (level 0) is omitted since it is always a "1". The first level -four "1"s- correspond to the four grey nodes in level 1 of the octree. In the second level, the first four values "0010" are the children of first "1" in level 1;

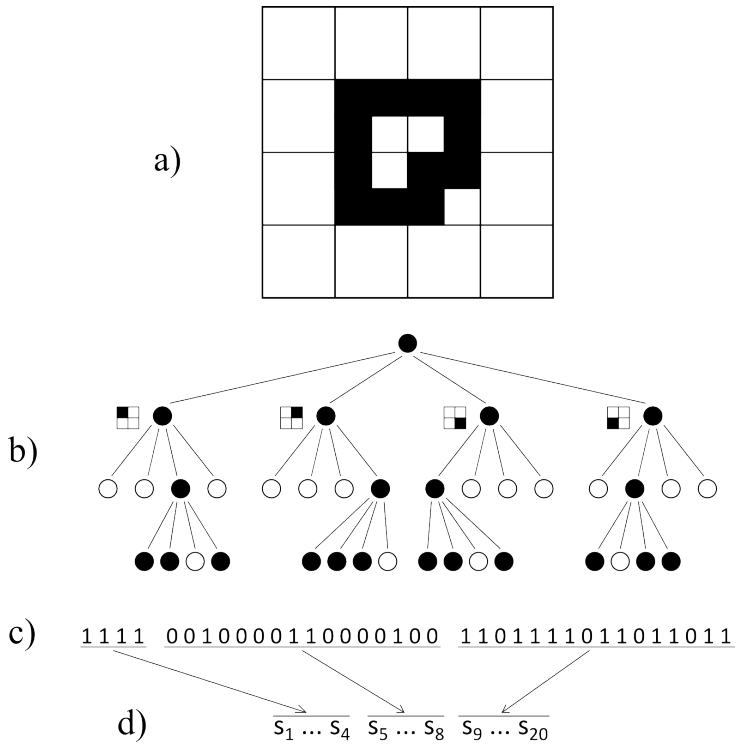


Figure 4.8: 2D example of a Surfel Octree representation. a) Shaded regions contain parts of the organ boundary and are represented by surfels. b) Octree structure and ordering of siblings during spatial subdivisions. c) Bit stream encoding the octree structure. Separate levels are underlined, and the root node is not represented. A total of 20 surfels are represented in the data stream, (d).

"0001" are the children of second "1", and so on. The first four values in level 3, "1101" are the children of the first "1" in level 2.

Surfels are arranged consecutively as they are found in a width traversal of the octree, (d) in the Figure. Note that this is the same traversal used to construct the bit stream, hence the order of "1"s and their surfels coincide. Of course, "0"s do not insert any data, so surfels are compacted without gaps in between.

The link between a node (a "1" in the bit stream) and the corresponding surfel is as follows: if the "1" is the i-th "1" in the bit stream, the surfel occupies the i-th position in the surfel stream. No auxiliary structures nor pointers are required.

The total amount of memory required for the representation of the tree

structure of Surfel Octrees can be bound under the assumption that the fractal dimension of the organ boundaries is 2 (in other words, we assume that the organ surfaces are not fractals). Let us note by v_k and g_k the number of Void and Grey nodes at the octree level k, $k = 0..n$. The number of surfels s_n at the highest resolution is obviously $s_n = g_n$. We know that $v_k + g_k = 8 * g_{k-1}$ for all k and that g_k is similar to $4 * g_{k-1}$ for the deepest levels of the octree, the first equation coming from well-known octree properties and the second one from our assumption that organ surfaces are not fractals, [191] (box counting for the estimation of the fractal dimension D is based on the well-known equation $g_k = 2^D \cdot g_{k-1}$). By using these two equations and adding the contribution of the different octree levels, we obtain that $v_k \simeq s_k$ for k values close to n. Then, the total number n_n of octree nodes can be approximated as $n_n = 2 * s_n + s_n/2 + s_n/8 + s_n/32 + ... < 2.688 * s_n$. Taking into account that our representation requires one bit per node to represent the octree structure, we can conclude that, in well-formed organs, the overhead of the proposed octree structure representation is less than three bits per surfel at the highest resolution. In other words, the octree of an organ like the Calcaneus having 90058 surfels at the deepest resolution (level n) requires 90058*3=270174 bits which are equivalent to 34 KBytes.

4.5 Conclusions

This Chapter has been dedicated to describe the hierarchical structure that stores and complements the geometry of Constrained Surfels.

Constrained surfels are used to represent organ boundaries. They are created from the samples in the model, but the discretization in the acquisition of samples introduces an inaccuracy on the exact position of the isosurface. A stick is a geometric element extracted from the model that delimits that inaccuracy and defines a local restriction to the isosurface.

Surfel Octrees are data structures that partition the space occupied by the model. Their leaves coincide with voxels. High-resolution surfels are assigned to the leaves while inner nodes are recursively populated with surfels that are built from simplifications of the surfels in their children. The orientation of the new surfels is computed as an average of its descendants and their position is adjusted to preserve the volume determined by their children.

Surfel Octrees are hierarchical multiresolution data structures. They therefore support general view-dependent visualization algorithms with dynamic fronts.

All data structures (octree and surfels) are stored and transmitted using a compact representation. Constrained Surfels are encoded using a CCPP for

the supporting plane plus a texture index compressed to only 4 bytes. The tree structure of Surfel Octree is also efficiently encoded using a scheme that results in just 1 bit per node, or less than 3 bits per surfel.

We can conclude that his new representation allows a progressive transmission of the organs with low memory footprint for both Constrained Surfels and the Surfel Octree.

Chapter 5

Interactive Inspection in Client-Server Systems

The ultimate goal of the present work is the definition of a scheme of a functional and interactive Atlas in low-cost portable client devices. The definition and construction of the data structures of the model has been presented in previous Chapters, while the present Chapter is devoted to the efficient visualization of volume data, the efficient data transmission between the Server and the Client and the set of int5eractive tools provided to the user. A first version of this work was published in [194].

To illustrate the methods described in previous chapters, the Server computer has been also used as a Client. This allows more accurate comparisons of the different techniques and the use of examples, containing larger organs and/or more organs present in a scene. Anyway, in order to prove the efficacy and usability of the proposed method, some examples and tests have also been performed in a low-cost tablet device.

5.1 Inspection of the inner structure of organs

In Chapters 3 and 4 a hierarchical strategy based on surfels for the visualization of organ boundaries has been presented. The result of this strategy is an Octree Forest that contains the Surfel Octrees of the organ boundaries of the model. This data structure is constructed in a preprocess phase and then it is stored on the Server.

However, a problem may arise from the well-established fact that medical experts are used to examining the interior part of the body with planar sections of it. X-ray images from late 19th century and, more recently, sonography, magnetic resonance imaging (MRI) and computerized tomography (CT) scans,

support this statement. We propose a strategy where clients can visualize planar sections of the initial volume model. In this method, sections are sent only in specific instants of the interaction.

During the inspection of the model, users may be interested in exploring the interior structure of the organs shown in the scene. In this case, users can define a section or clipping plane (orientation and position) to be sent to the Server. This request is processed by intersecting the plane with the model volume. For every voxel, the nearest sample point to the plane is projected to it. The set of projected points, that may be associated to different organs, is stored as an image. The Server sends the image back to the Client and this image is rendered as a texture in the section plane. The renderer discards geometry placed in front of the plane -in the direction to the observer-, thereby the user can inspect the planar image without obstacles.

Since the scene contains only selected (by the user) organs, the projection algorithm includes pixels of points associated with the same organs. This situation may produce an image with void areas -where non-selected organs would be projected-. Fragments corresponding to those pixels are rendered in the Client as transparent ones, so they do not occlude organs behind the plane.

No more data transference is required after the arrival of the image. Subsequent user inspection of the model and interaction is performed entirely on the Client device until a new section is requested.

The complete process of the inner structure inspection with the interactive inspection tool is outlined in Figure 5.1. In the first stage the user defines the position and orientation of the clipping plane and the Client and send this request to the Server. There, the plane is intersected with the model and the corresponding image is generated. This image is sent back to the Client where is transformed into a texture. In the second stage, the scene is arranged to include the texture: surfels lying in the negative half-space of the plane (closer to the viewer) are discarded and the texture is placed in the correct position and orientation. If solicited, the saturation of the texture is changed. After these tasks, the user interacts locally with the scene, not involving further communication with the Server.

Four examples are presented to illustrate this tool (Figures 5.2, 5.3, 5.4 and 5.5). Boundaries of the texture are outlined in white. In the left and right column, the same section is presented with different views. The rows illustrate the ability to change the saturation of the texture to highlight the section. The upper row uses the original color, in the middle row the texture is slightly de-saturated and in the lower row the texture has only grey tones. All four

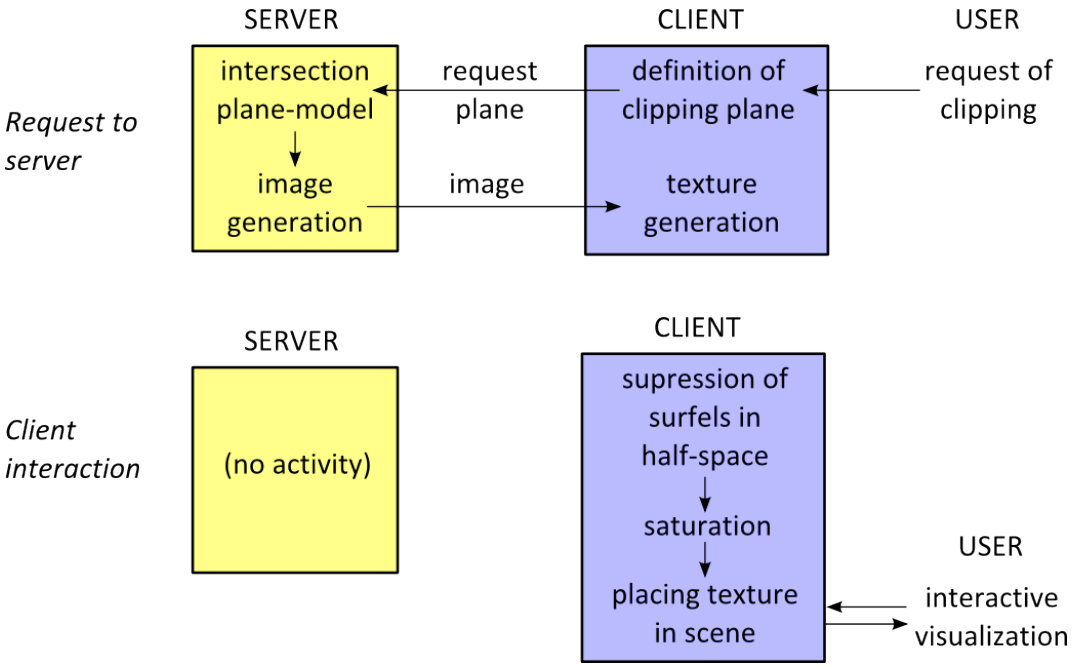


Figure 5.1: Stages of the clipping process: Request to the Server and Client interaction.

figures have the same configuration of views and color settings.

Data about the images transmitted are presented in Table 5.1. The second column is the resolution of the image expressed in texels. The third column is its size in bytes. Different compression algorithms can be applied to reduce the size of the data transmitted.

Organ	**resolution**	**size**
Backbone	433 × 617	389431
Knee	521 × 191	205105
Kidney	256 × 286	124392
Heart	162 × 440	94618

Table 5.1: Size of sections of Figures 5.2, 5.3, 5.4 and 5.5. Resolution is expressed in texels and size in bytes.

Figure 5.2: Section of the backbone. Left and right columns differ in the view direction. Upper row shows the section in real color. In the middle row the color is slightly de-saturated. Bottom row offers the section in grey tones.

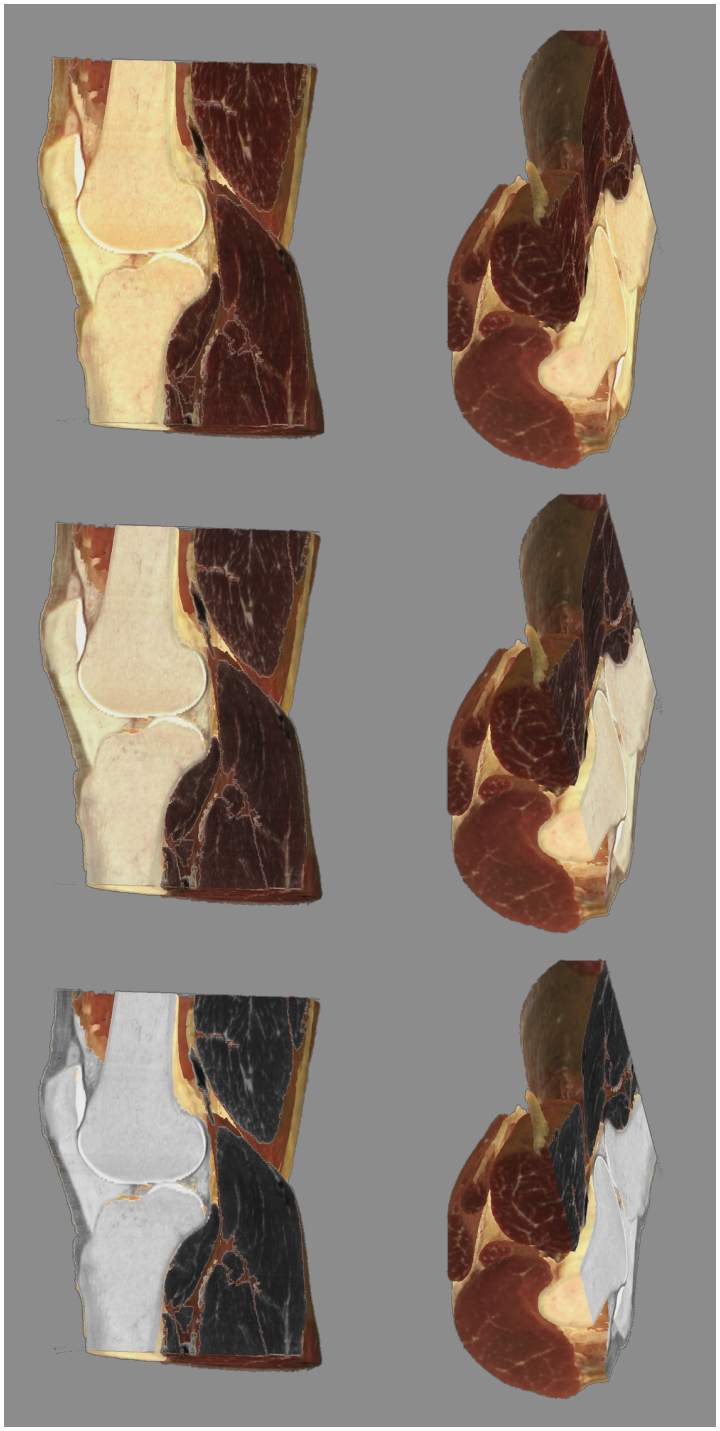


Figure 5.3: Section of the knee. Left and right columns differ in the view direction. Upper row shows the section in real color. In the middle row the color is slightly de-saturated. Bottom row offers the section in grey tones.

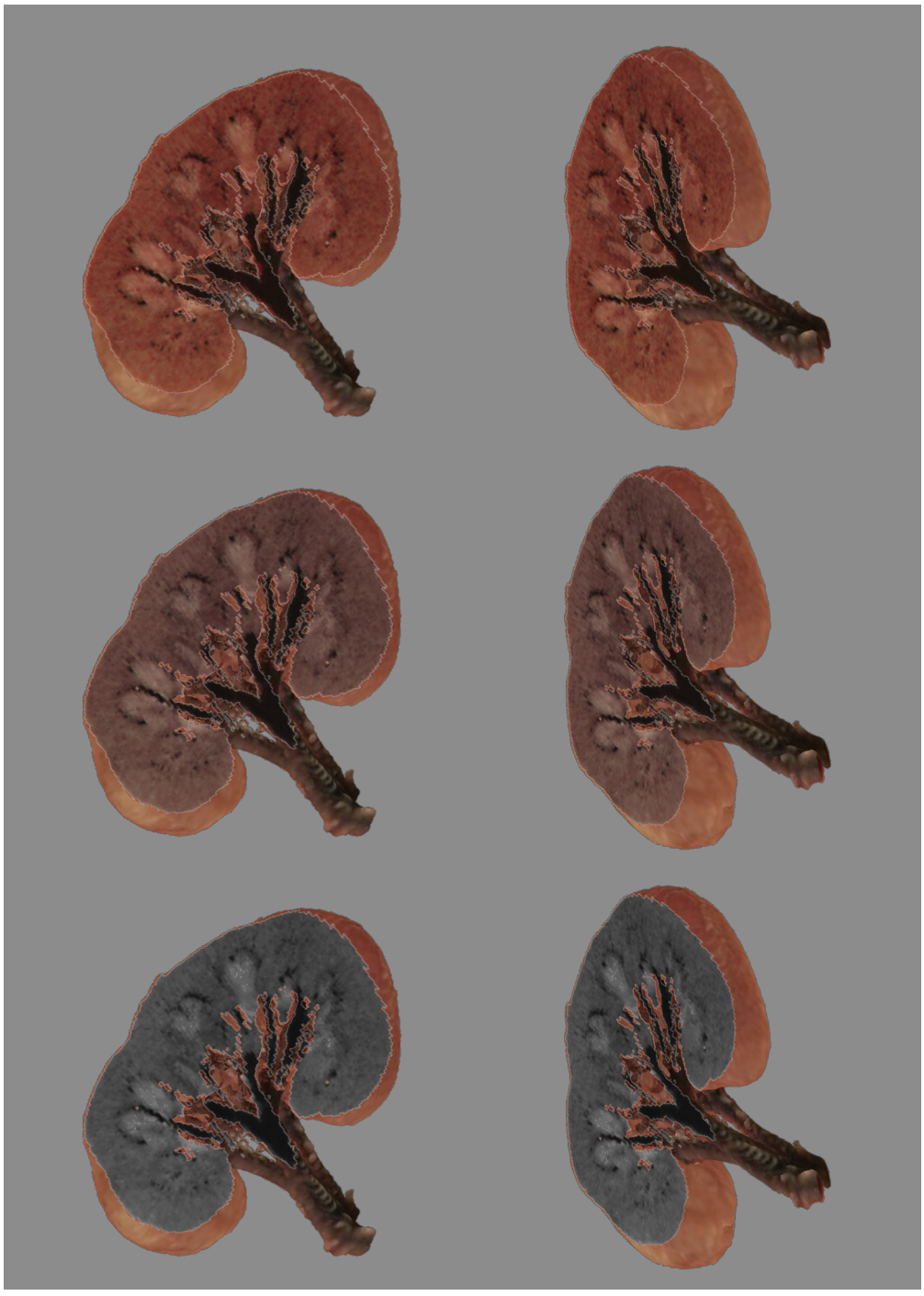


Figure 5.4: Section of the kidney. Left and right columns differ in the view direction. Upper row shows the section in real color. In the middle row the color is slightly de-saturated. Bottom row offers the section in grey tones.

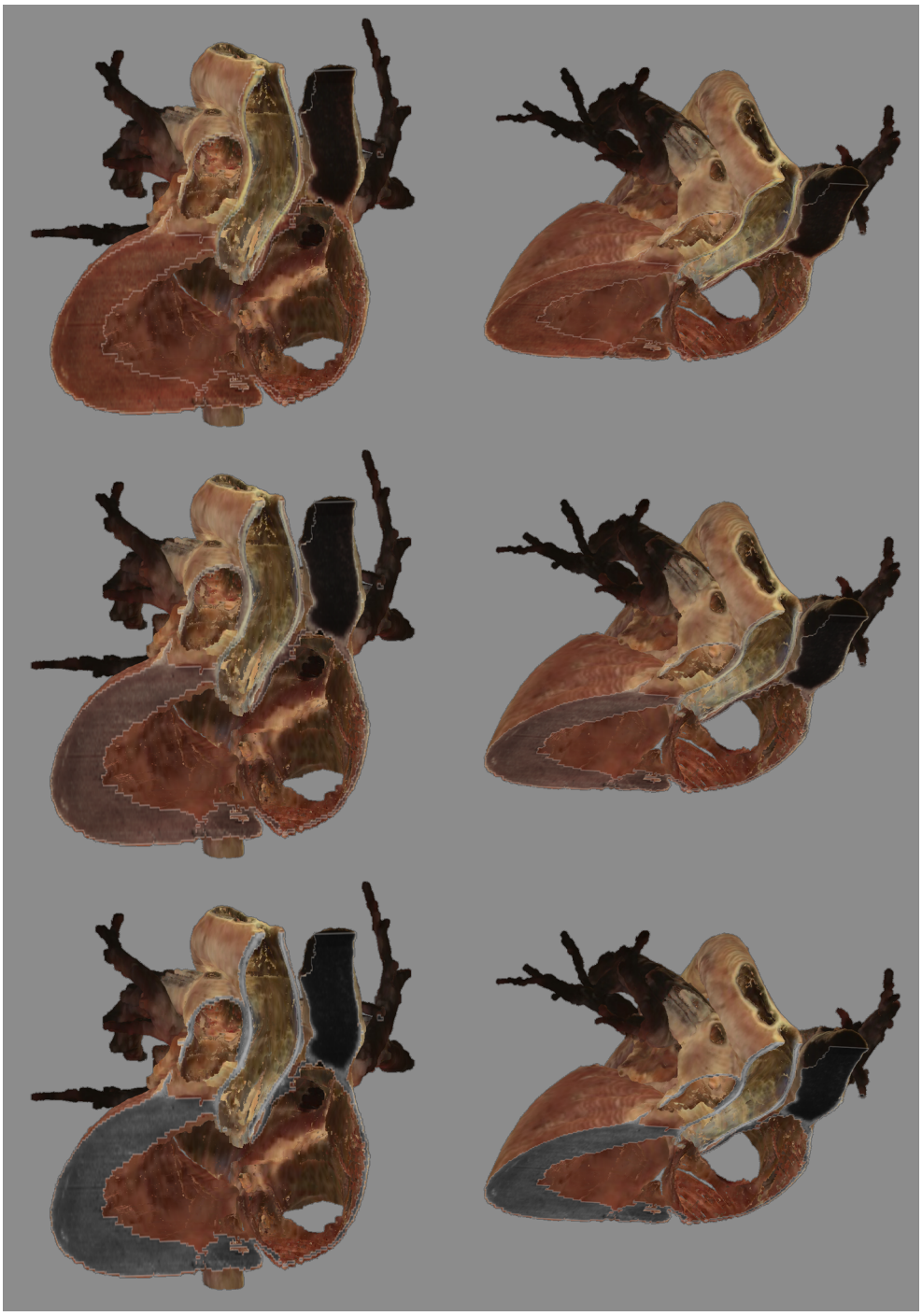


Figure 5.5: Section of the heart. Left and right columns differ in the view direction. Upper row shows the section with in color. In the middle row the color is slightly de-saturated. Bottom row offers the section in grey tones.

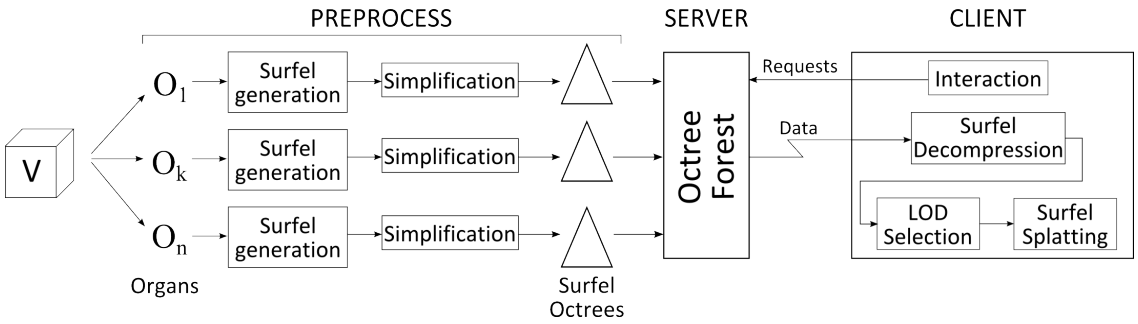


Figure 5.6: Overview of the presented approach. The preprocess is done for every organ. The set of Organ Octrees become the Octree Forest.

5.2 Data Transmission and Rendering

Figure 5.6 presents the overall architecture of our proposal. The algorithms starts from a segmented binary volume model V without density information. In binary segmented visualizations, every voxel stores an index to the organ in its specific location. In our case, we also assume that voxels in V contain the color attribute of its tissue. In a preprocess, organs in the region of interest are detected and a Surfel Octree is computed for each organ. The Server stores the set of all organ octrees -the Octree Forest- and sends parts of it to the clients based on their interactive requests.

The computation of Surfel Octrees is repeated for each organ in the region being studied. For each octree, surfels in the leaf octree nodes are first computed and smoothed. In a second step, grey octree nodes are computed by a bottom-up simplification process. The Surfel Octree generation algorithm is detailed in Chapter 4.

The Octree Forest is rather compact and can be stored in the Client. It is represented as an array of Surfel Octrees, each of them being indexed by its organ label. Transmission to the clients is performed on demand and in a progressive way. Every client c has a lower resolution version of the Octree Forest, as represented by its resolution array $R_c = (d_{c,1}, d_{c,2}, ...d_{c,n},)$, where $d_{c,k}$ is the depth of the octree corresponding to organ k in the local representation of client c. The server stores a two-dimensional array containing the resolution arrays of all clients.

Surfel rendering in the client devices is based on a preorder traversal of the Surfel Octrees of the Octree Forest. The octree traversal keeps track of the current depth in the octree and the path from the root, since the 3D location and size of the cube of any visited node can be easily computed from the

depth and path information. The geometry of individual surfels is created on the fly by decoding the Connected-Cubes plane information, transforming this local plane to the World coordinate system and rendering an hexagon that circumscribes the polygon of intersection between the surface plane and the cube of the octree node. A specific fragment shader creates a circular surfel by removing hexagon fragments outside the circle and assigns colors to fragments by projecting the surfel indexed texture.

5.3 Interactive Inspection Tools

We start interactive sessions by defining an inspection region, obtaining its list of organs and sending this list and the octree structure of the Surfel Octrees of the corresponding Forest to the clients. Our results, as shown in the following Section, confirm that the amount of involved information is small enough for interaction purposes.

The user selects a group of organs to inspect from the organ list, and the client sends requests for each of these organs to the server, which in turn sends the set of textures of each organ and a set of surfels corresponding to a low resolution octree level. In our present implementation, assuming that a certain Surfel Octree has depth equal to n, we initially send all surfels at octree level $n-2$.

Progressive octree transmission is performed on demand. Requests for increased detail (surfels at levels $n-1$ or n) can be issued by the client at any moment during the interactive inspection session. When a client with a resolution array R_c sends a query for a refinement of a certain organ k, the server increases $d_{c,k}$ by one and it sends the next octree level for organ k from its Octree Forest to this client along with compressed surfels. Client queries include the requested level of the corresponding organ octree to avoid potential synchronization problems. The client receives it and also increments its $d_{c,k}$ by one.

Observe that textures and octree structures are LOD-independent: they must be sent only once per organ to the clients. Memory requirements of textures and surfels at different octree levels are discussed in the next Section.

During interactive sessions, users can select groups of organs, add or remove some of them at any time, and can decide to inspect the whole volume model at a low resolution or ask for a better resolution in certain specific organs. Since the whole structure of the Surfel Octrees is available at the client side, rotation and zooming operations can be autonomously performed in the client without any further transmission from the server. If a region of the model needs to be detailed, surfels from lower levels can be displayed on demand.

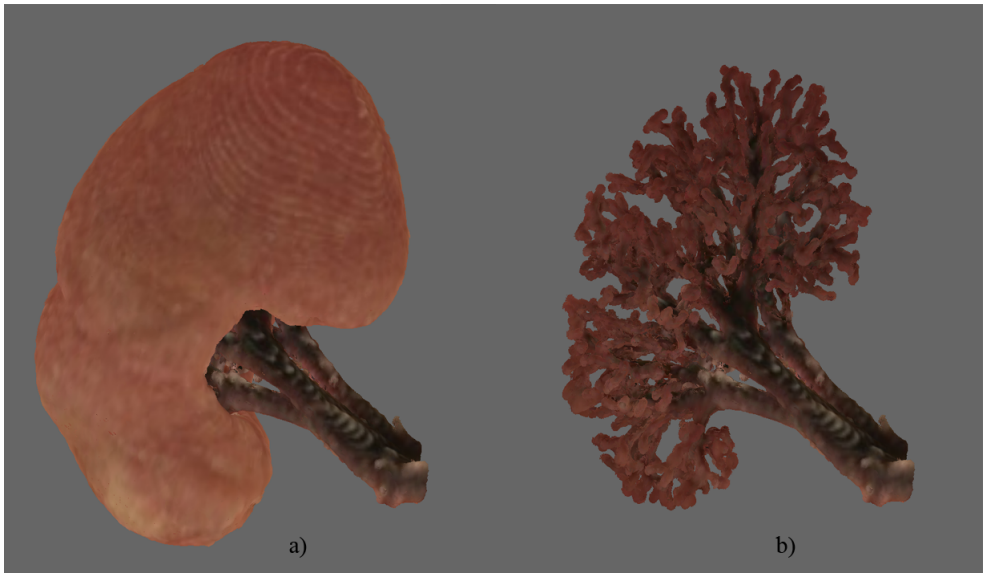


Figure 5.7: Interactive selection of organs. a) Scene with the Kidney and the Renal Artery. b) The Kidney has been interactively selected out, leaving the Renal Artery as the only organ in the scene.

At any moment, a Client stores a set of multiresolution Surfel Octrees with resolutions given by its present resolution array. Interaction exploits these multiresolutions by supporting a number of different paradigms:

- Organs can be individually selected and made either visible or invisible to highlight the remaining structures. The rendering algorithm simply traverses the subset of visible organ octrees. In Figure 5.7-(a) The Kidney and the Renal Artery are both visible, whereas in 5.7-(b) the Kidney has been interactively removed, showing the complete structure of the Renal Artery.

- Visible organs can be assigned with specific and different octree resolutions. Organs at lower resolutions will be rendered with coarser surfels, which correspond to higher levels in their Surfel Octrees.

- The user can use a *precision inspection sphere* and move it through the volume. The sphere acts as a "resolution magnifying lens", where all surfels in the sphere are rendered at the maximum resolution available in the client at that moment. The octree level for each surfel is determined by the resolution of the corresponding organ and by detecting whether

every surfel fragment is inside the higher resolution sphere or not. The sphere defines a dynamic multilevel front in the Surfel Octree. This method requires that the two levels of the organs being scrutinized by the sphere must be loaded in the Client memory at the same time.

- Distance-based view-dependent multiresolution is also possible. In this case, the level of the surfel in the octree is computed from its distance to the observer, like in other view-dependent algorithms. Two major strategies may be applied in the determination of what surfels change their resolution level. The simplest one consists in the change of the whole organ simultaneously when a specific point or surfel of the organ reaches a threshold distance. In the second one, the change of resolution is determined per voxel, resulting in organs represented with multiple levels. This strategy also requires that several levels of the organ must be loaded in the Client memory at the same time.

- Volume inspection is available at any moment through textured planar sections, as described in Section 5.1. Rotation and zooming operations with these planar sections can also be performed autonomously in the client, as discussed in Section 5.4.

5.4 Results and Discussion

Examples of different regions of the Visible Human model have been implemented to test hypotheses and to decide on alternatives. Some of them are presented in this Section with the conclusions that have been obtained from their analysis.

All images have been generated by a portable computer with an Intel Core i7 CPU with 4 cores at 2GHz, 4 Gbytes of RAM and an NVidia GeForce GT550M GPU with 2 Gbytes of RAM. The same machine performs the preprocessing and the Server part.

5.4.1 Results using surfels with color intensity gradients

Figures 5.8 and 5.9 have been obtained using the Gradient of Intensity color method.

Figure 5.8 shows a rectangular region of 75 x 67 x 50 mm. representing part of the neck. In this case, the total number of processed organs is 49, resulting in an Octree Forest of 49 components. Figure 5.9 shows a rectangular region of 67 x 94 x 80 mm. representing part of the head. In this case, the total number of components in the Octree Forest is 32. Resolutions of the model

regions presented in figures 6 and 7 are 225 x 200 x 150 voxels and 195 x 280 x 240 voxels respectively.

Rendering and interaction snapshots with these two Octree Forests are presented in Figures 5.8 and 5.9. In both Figures, the first column shows the volume with all organs, while conjunctive tissue has been removed in the second column for the sake of clarity. It should be observed that users can turn organs on and off in a direct way to have either a global view or to inspect a selected subset of them. Second and third rows in Figures 5.8 and 5.9 present several interaction possibilities. Users can interactively select the proper resolution level for every organ to highlight some of them at the maximum resolution, selected organs can be rendered with semi-transparent surfels, and a high-resolution sphere can be properly located to inspect some organs at a maximum detail. A short video has been recorded of these scenes and can be viewed in `http://www.cs.upc.edu/~jsurinach/Thesis/Videos.html`, video 4.

Rendering all organs in Figure 5.8 at the maximum resolution requires a total of 1.89 Msplats, while rendering them at resolution level $n-3$ requires a total of 23 Ksplats. In the case shown in Figure 5.9, the number of splats is 1.81 Msplats and 22 Ksplats respectively.

We start the progressive transmission of the organ octrees at resolution level $n-3$ and then successively send the next levels on demand until the maximum resolution surfels at level n have been sent. In our two cases, level $n-3$ contains 23 Ksurfels and 22 Ksurfels, as already mentioned. With a compression rate of 11 bytes/surfel, this is equivalent to 253 KBytes and 242 KBytes respectively, which give an acceptable quality with a reduced data flow. Organ octrees are then progressively sent to the clients and refined, and some of them will finally reach the maximum resolution level n. The transmitted data flow depends on the subset of selected organs for high resolution rendering. The worst case (transmission of all maximum resolution surfels) requires 1.89*11= 20.8 MB in the case depicted in Figure 5.8 and 1.81*11= 19.9 MBytes in the case in Figure 5.9.

The overall application runs at interactive rates with a frame rate always above 30 fps. Transmission times are reasonable and compatible with real-time interaction. In the examples in Tables 1 and 2, the amount of transmitted information is 0.25 MBytes for level $n-3$ and 1 MByte for level $n-2$, resulting in transmission times between 1 and 4 seconds in our tests. The maximum resolution requires the transmission of 20 MBytes (around 50 seconds depending on Internet conditions) but it can be performed in parallel with real-time inspection. Users can interact with models at a resolution level of $n-2$ (or $n-1$) while the full resolution level is being received. Observe that network bandwidth limitations do not significantly affect the frame rate, as several res-

olutions of the inspected organs always reside in the client memory ready for inspection and interaction.

Computing times both in preprocessing and transmission are proportional to the number of surfels at maximum resolution, which is a small fraction of the volume resolution (between 14% and 28% in our experiments). This fraction decreases as resolution increases, based on the properties of boundary-encoding octrees.

5.4.2 Discussion of surfel plane encoding

Section 3.2.5 presented the possibility of different encodings of the planes that support the surfels. The encodings differ in their memory consumption and visual appearance. We present two examples, Figures 5.10 and 5.11, to illustrate the visual effect of these plane quantizations.

As stated in sections 3.2.5 and 3.4, the information stored in a surfel must include:

- The Connected Cubes parameterization of the plane of the surfel: three coordinates and the id of the C-cube.
- The index of the associated texture, using the Projective Texture Mapping method.

There are eight different C-cube configurations, therefore only three bits are needed for their encoding. Twenty image textures are generated per organ and they are encoded with five bits, giving room for twelve additional images when surfels are occluded, as described in section 3.3.4. Only the three coordinates allow a scheme of quantization. Three options are considered:

- **Four bytes per coordinate**. This is a usual quantization of scalars, representing a no-compression scheme and the Ground Truth image. For simplicity, the C-cube and the texture index are encoded in one byte each. The overall memory occupation of a surfel is 14 bytes/112 bits. They are the organs labeled a) in both Figures.
- **Ten bits per coordinate**. Including the C-cube (3 bits) and the texture index (5 bits plus 2 bits of padding to obtain a byte-alignment size), the surfel occupation is 5 bytes/40 bits. Label b) in the Figures.

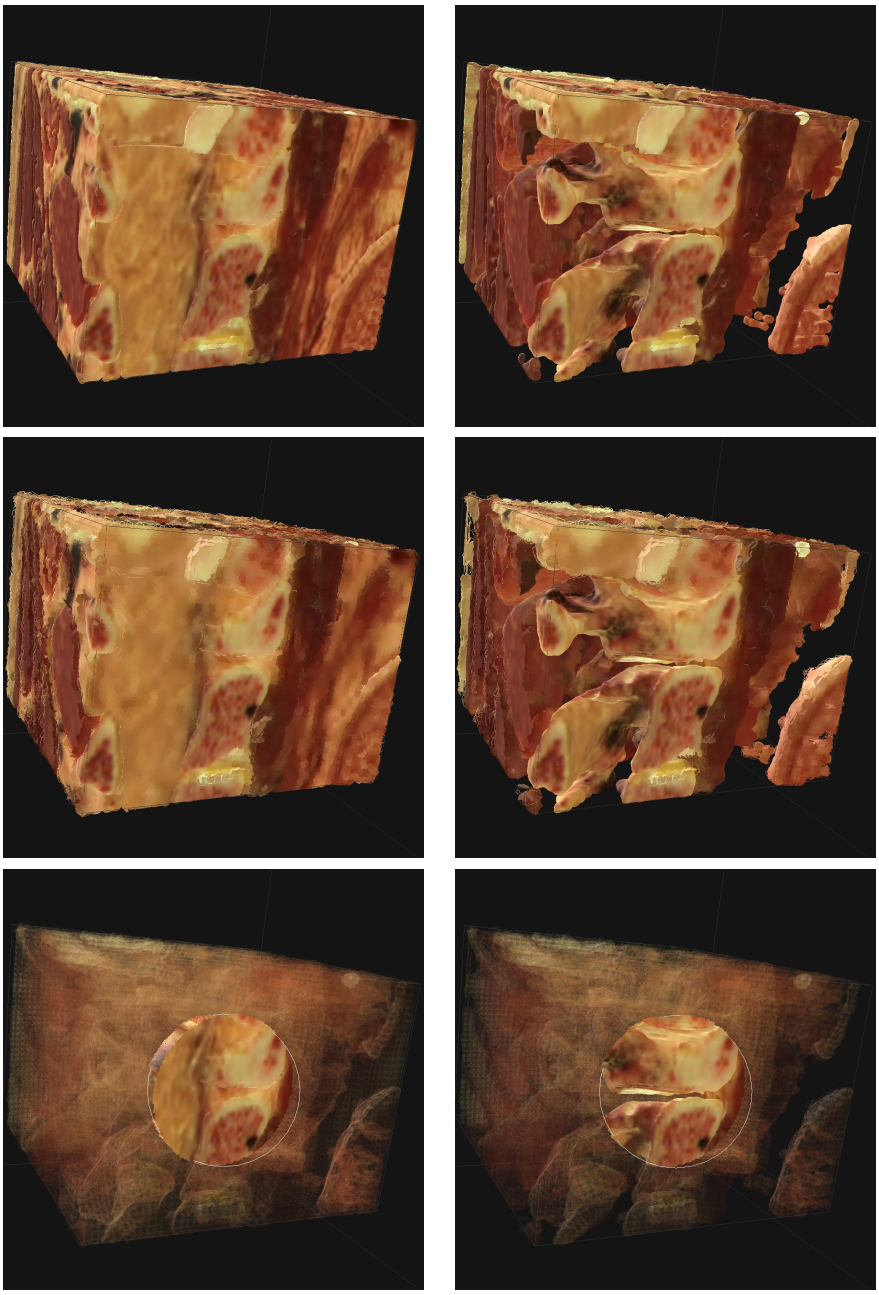


Figure 5.8: Left part of the neck. Left column shows all organs while in the right column, only a subset of the organs have been selected. In the top row all organs are displayed at maximum resolution. In the middle row bones are shown at maximum resolution while the rest of organs are displayed with surfels from upper levels of the octree. In the bottom row the effect of the Precision Inspection Sphere is shown

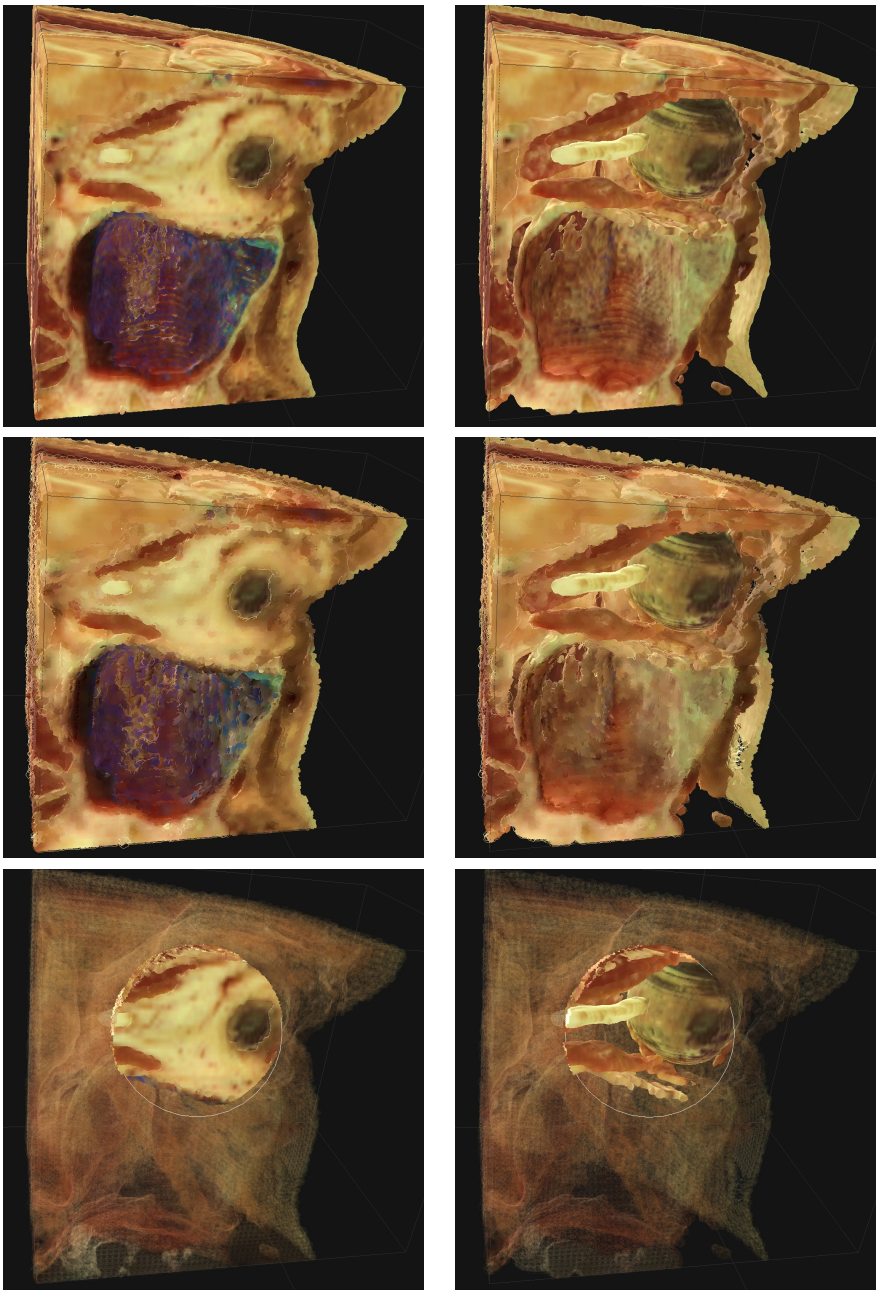


Figure 5.9: Left part of the head. Left column shows all organs while in the right column, only a subset of the organs have been selected. In the top row all organs are displayed at maximum resolution. In the middle row the eye and the optic nerve are shown at maximum resolution while the rest of organs are displayed with surfels from upper levels of the octree. In the bottom row the effect of the Precision Inspection Sphere is shown.

Figure 5.10: Heart and some vessels. Comparative between plane quantizations: a) 112 bits, b) 40 bits, c) 32 bits.

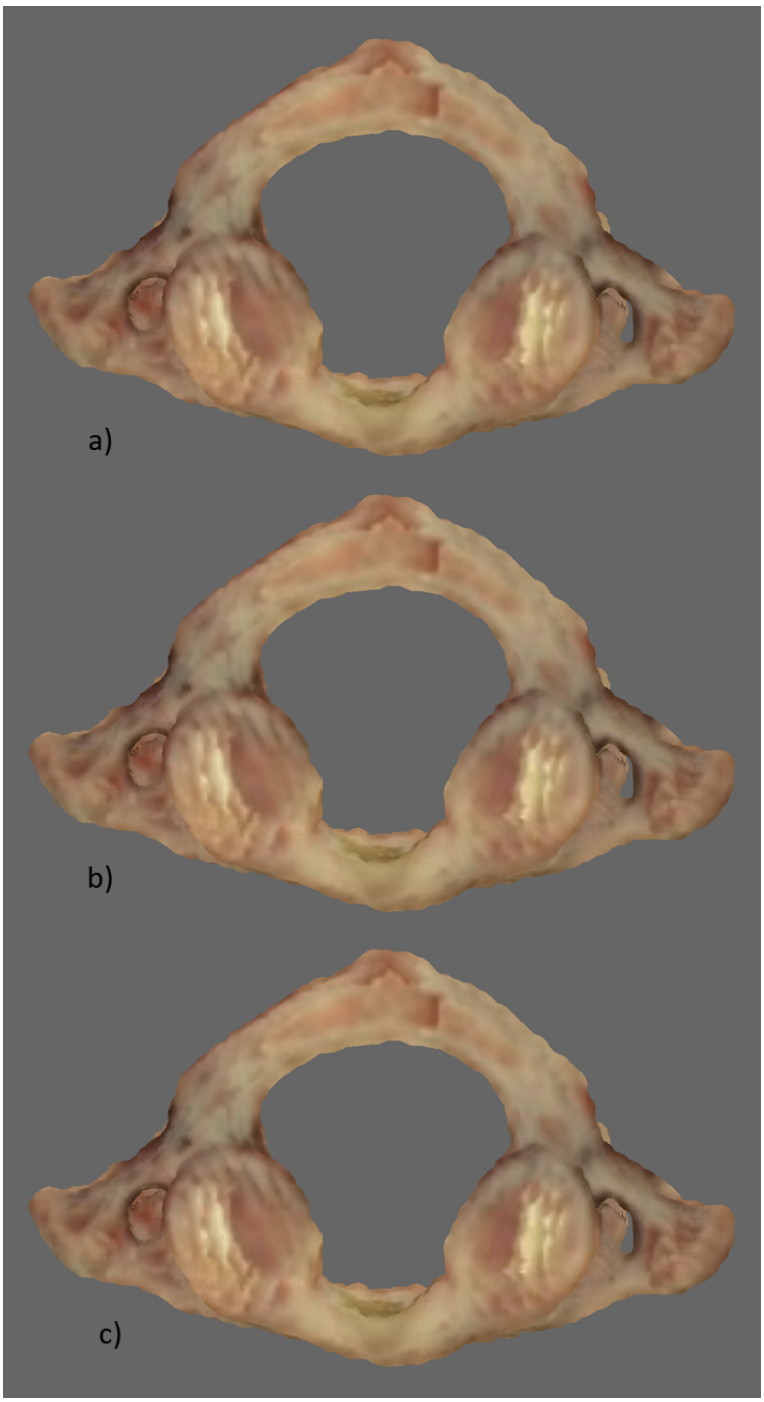


Figure 5.11: Atlas. Comparative between plane quantizations: a) 112 bits, b) 40 bits, c) 32 bits.

	atlas	**heart**
112-40	278.7697	273.4695
112-32	266.8662	260.5573

Table 5.2: PSNR values for the 40 and 32 bit encoding of the atlas and the heart Figures 5.11 and 5.10, tested against the 112 bit encoding.

- **Eight bits per coordinate**. In this situation the occupation is 4 bytes/32 bits. Label c) in the Figures.

We must take into account that the 32 bits encoding consumes 20% less memory than the 40 bits encoding. In the heart model that means a savings of 1.3 Mb.

To compare both encodings, the HDR-VDP-2 visual metric from Mantiuk et al. [183] has been used. In this metric, an image of the scene is taken and is compared to a reference image. Test images are obtained from 40 and 32 bits encoding scenes, whereas the reference image is from the ground truth 112 bits scene. Visual differences perceived by the human eye are highlighted in the resulting images. Results are shown in Figures 5.12 and 5.13. Reddish points correspond to areas visually more different to the reference image while bluish ones are more similar. We can appreciate that the 40 bits quantization image has lesser red points than the 32 bits one, as expected.

Moreover, the PSNR -Peak Signal-to-Noise Ratio- values of the test images compared to the reference image are also presented in Table 5.2. Since PSNR is inversely proportional to the Mean Squared Error, the higher the value, the better. Again, images of the 40 bit encoding offer the best result.

However, a visual exam of the original figures -along with other examples carried out- presents almost indistinguishable images in the three encodings analyzed. Therefore, the use of the quantization of 32 bits is totally justified and is used in the remaining of this work. With this encoding, a plane is successfully compressed in 27 bits.

An encoding of 7 bits per coordinate is not considered since its occupation would be $7 \times 3 + 3 + 5 = 29$ bits, which fit again in 4 bytes (with 3 padding bits to obtain a byte alignment). This is the same occupation and presumably it will produce a worse visual result, therefore this option is discarded.

Considered globally, the usual definition of a plane (4 scalars $\times$ 4 bytes per scalar) plus one byte for the texture index, produces a result of 17 bytes

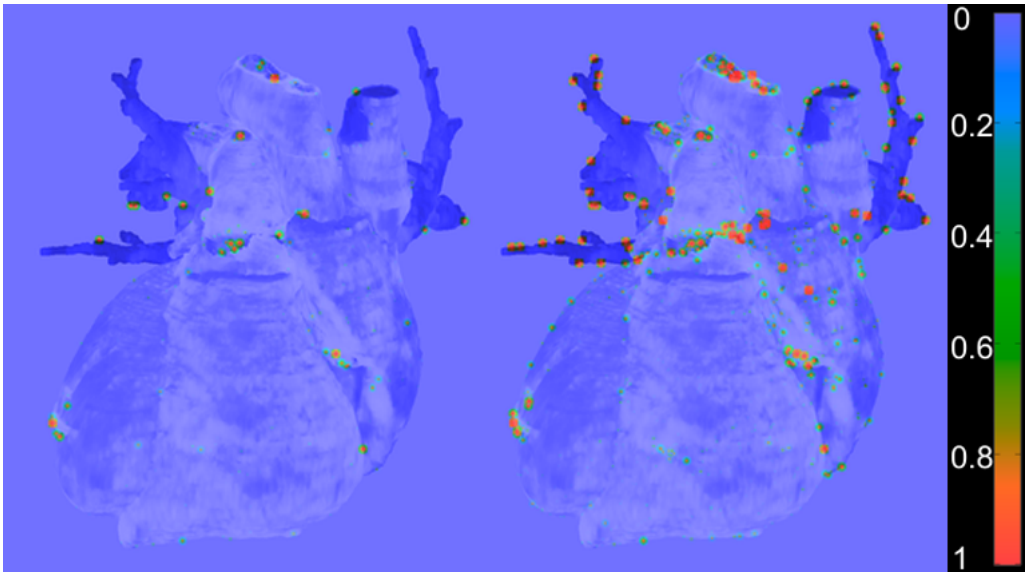


Figure 5.12: HDR-DVP-2 visual metric of plane quantizations using the heart image 5.10. The reference image is the one with 112 bits. At left, the test with 40 bits and at right, with 32 bits.

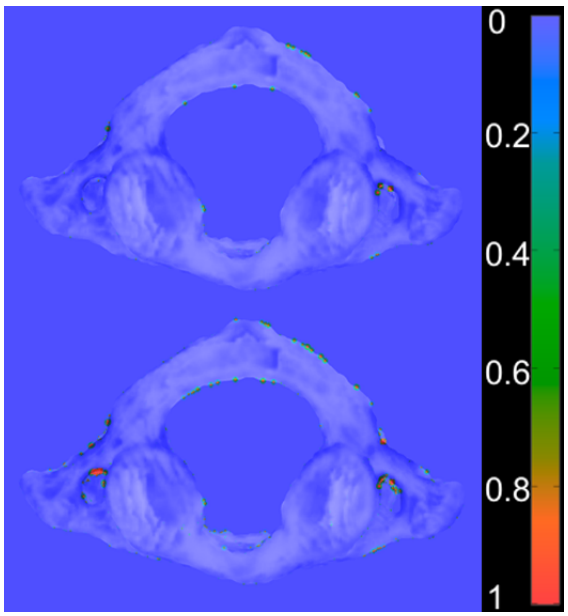


Figure 5.13: HDR-DVP-2 visual metric of plane quantizations using the atlas image 5.11. The reference image is the one with 112 bits. At the top, the test with 40 bits and at the bottom, with 32 bits.

per surfel. Our proposal, 4 bytes per surfel, represent the 76.47% of memory reduction.

Observe that Property 2 (Section 3.2.4) ensures that plane representations with 8 bits per coordinate are faithful in a 128 x 128 x 128 subdivision of any octree node. This is subpixel precision, provided that octree nodes project in less than 128 pixels in screen space, which is always the case. This explains the good results in Figures 5.10 and 5.11.

5.4.3 Discussion of surfel textures encoding

The network bandwidth occupation can be decreased by reducing the amount of transmitted data. Before the interactive session, the Server sends images of all selected organs of the scene to be used as textures. In our approach, there are 20 images at 128×128 pixels of resolution, and 26 bits of color depth per organ. This results in 1064960 bytes, slightly over 1 Mb. We have tested scenes with roughly one hundred organs, making this initial transmission very expensive, even though it is done only once. To reduce this amount of data and with the purpose of being faithful to the model, a lossless and a lossy compression modes (png and jpeg formats) are tested. In Figure 5.14 four images of the same scene are compared. Upper row images are in png format while the lower ones are jpeg. The resolution of left column is 256×256 and that of the right column is 128×128. A third tested format, bmp, is not shown because the images are identical to the png ones, since png is a lossless format. The similarities between upper and lower rows are noticeable, making the png and jpeg images almost indistinguishable. Differences arise between the left and right columns, where a halved resolution causes a light blurring on the organ. Table 5.3 shows the memory consumption of the four combinations plus the bmp format. Beyond the visual similarities of the png and jpeg formats, jpeg offers lower occupation. Moreover, the jpeg 256^2 image gives better resolution than the png 128^2 with 40% of its size. Other examples performed show similar results (both in visual appearance and memory consumption) to this case.

5.4.4 Interaction with Surfel Octrees

The following examples use the Left Foot model. Forest Octree information about this scene is given in Table 5.4. Only 25 out of the 96 organs shown in the scene are represented in this Table. The exposed columns are: the number

	256×256	**128×128**
BMP	3.932.214	1.966.188
PNG	1.850.643	631.628
JPEG	256.616	100.380

Table 5.3: Memory consumption in Bytes of images in figure 5.14 plus the bmp format as a reference.

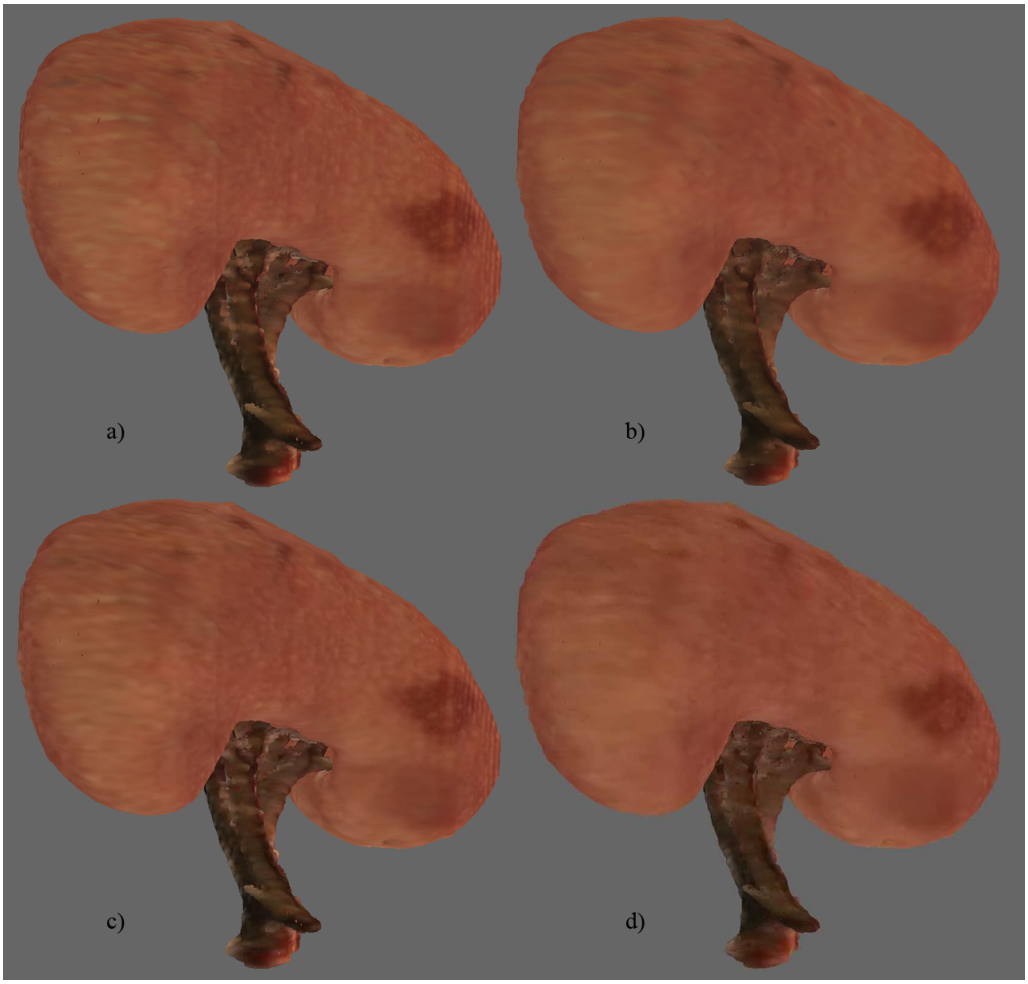


Figure 5.14: Comparative between formats and resolutions of the images of organs. a) png, 256 pixels. b) png, 128 pixels. c) jpg, 256 pixels. d) jpg, 128 pixels.

Organ	**n**	**n-2**	**size**	**depth**
Calcaneus	90058	5519	83	8
Abductor Digiti Minini	70457	4136	98	9
Flexor Digitorum Tendon	44798	2404	141	9
Metatarsal 1st	43061	2565	58	8
Talus Cartilage	40730	2033	57	8
Interosseous Dorsal 4th	31234	1869	50	8
Flexor Hallucis Tendon	28958	1592	152	9
Cuboid	25759	1531	36	7
Lumbricals	25557	1454	56	8
Metatarsal 5th	24830	1504	61	8
Tendo Calcaneus	23024	1206	31	8
Interosseous Dorsal 1st	20116	1200	32	7
Extensor Hallucis Tendon	18968	997	110	9
Interosseous Plantar 3rd	14590	842	43	8
Cuneiform Intermediate	12464	754	28	7
Phalanx Proximal 3rd	6339	373	26	7
Metatarsal Cartil 5th	3647	222	89	9
Phalanx Middle 3rd	2763	168	13	6
Phalanx Distal 3rd	2117	120	10	5
Tibial Nerve	1911	107	7	7
Phalanx Distal 5th	1466	69	10	5
Plantar Nerve Lateral	629	35	6	5
Sesamoidal Cartilage	336	16	5	4
Phalanx Cartil Prox 2nd	125	4	5	5
Phalanx Cartil Mid 4th	26	2	2	4

Table 5.4: Information about of some Surfel Octrees (25 out of 96) of the foot image. Columns are: name, number of surfels at level n (finest) and $n-2$ (more coarse), size of maximum side of bounding box (in mm.) and depth of the octree.

Organ	n	n-2	size	depth
Kidney L	328350	18131	251	8
Renal Vein L	200995	10250	272	9

Table 5.5: Surfel Octree information for the Kidney images. Columns are identical to those of Table 5.4.

of surfels in levels n and $n-2$ of every octree, the width of the cubic space occupied by the root node -the voxel size in the Visible Human dataset is 1/3 mm.- and its depth. Organs are ordered by number of surfels. Notice how long organs (like tendons) have a relative small number of surfels whereas their Octrees are the biggest ones in size and depth. Table 5.5 with Octree Forest information of previous Kidney Figures is provided for comparison purposes.

Differences of surfels between levels of the Octree are highlighted in Figures 5.15 and 5.16. At the top of Figure 5.15 is a foot rendered at the most detailed level (n); at the bottom is the same foot at a coarser level ($n-2$). Zooms on the right illustrate that organ boundaries with small curvature ratios are better defined with surfels of more detailed levels. This undesirable effect appears when normal vectors of adjacent surfels differ in an appreciable angle.

A similar situation is depicted in Figure 5.16. The small curvature ratio originates from a boundary of the inspected region of the model. The cut in the model produces the right angle in the Tibia. The zoomed region on the left (surfels of level n nodes) exhibits a well-defined feature. On the right, individual surfels of level $n-2$ are clearly visible and overrun the cut. However, the planar boundary of the cut appears identical in both images because the normal vectors of these surfels are almost identical.

In Figure 5.17 there are several views of the left foot. All these images come from the same session to illustrate two interactive abilities. In particular, the capacity of adding and removing organs from the scene, Figures 5.17-(a)-(d). And the section tool, Figures 5.17-(e),(f), described in Section 5.1. The normal vector of the plane cut is parallel to the viewing direction from the observer; subsequent rotations produce the images shown in the Figure. Similar interactive sessions are displayed in videos 1 and 2 of `http://www.cs.upc.edu/~jsurinach/Thesis/Videos.html`.

The level of detail shown does not have to be the same in all organs. Each organ can be specifically displayed at any required precision level. It is also possible to simultaneously display two levels of the same organ using a multilevel front. To take advantage of this ability the client imple-

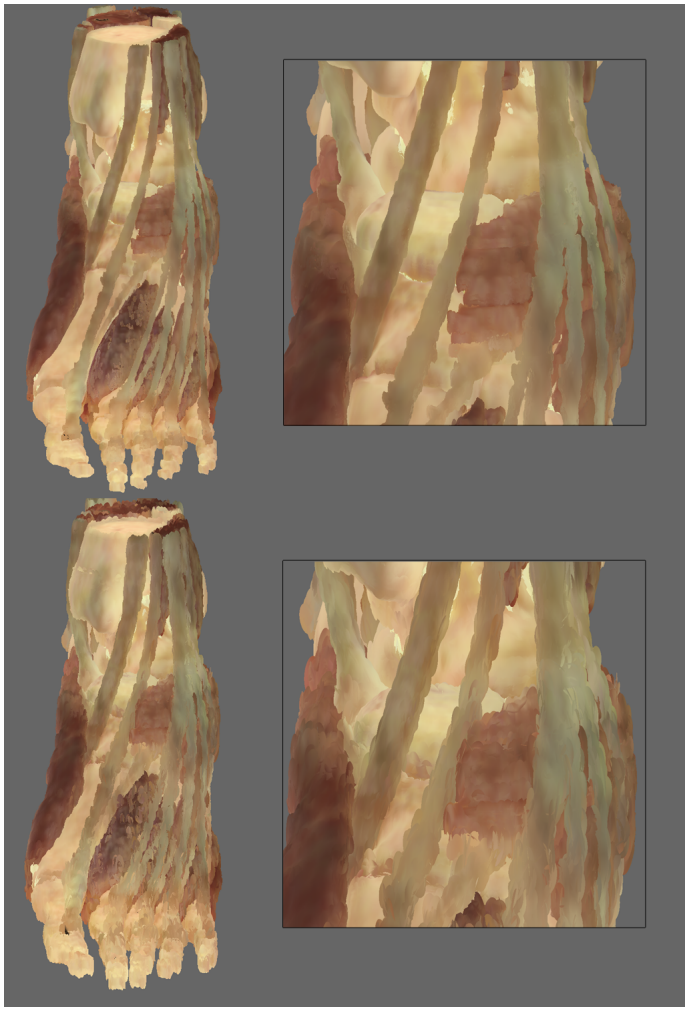


Figure 5.15: Top row: Foot at the most detailed level (n) and zoom of a region. Bottom row: The same but at coarser level ($n-2$). Individual surfels can be seen at the outline of the ankle at the coarser resolution (level $n-2$).

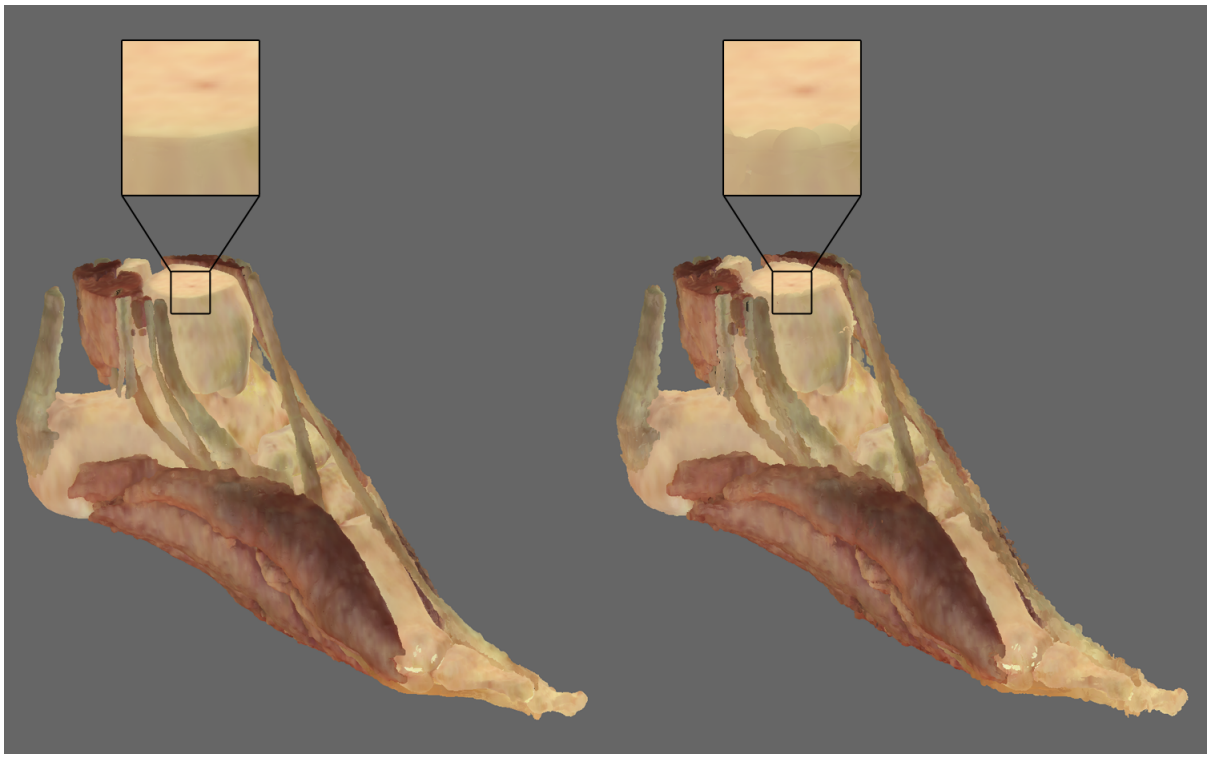


Figure 5.16: Detail of surfels from different levels of the octree at a boundary of the model. Left, foot at level n. Right, at level $n-2$.

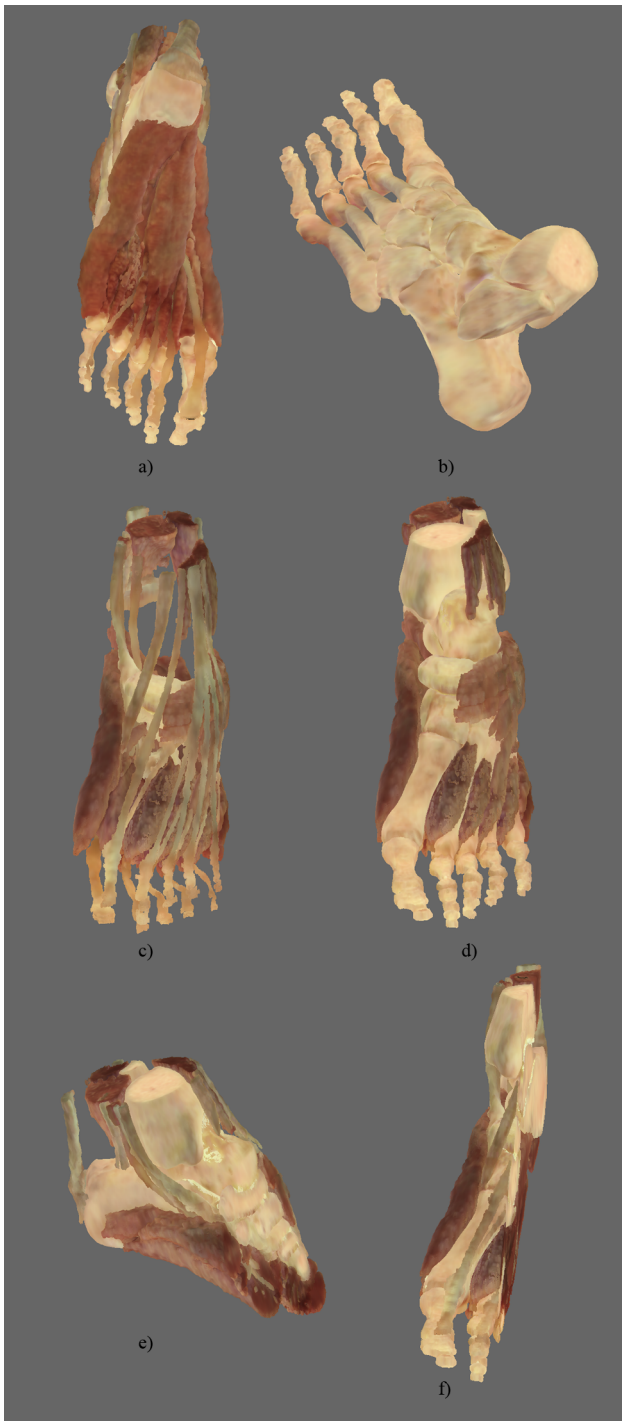


Figure 5.17: a) Plantar view. b) Only bones. c) Muscles removed. d) Only bones and muscles. e) Medial cut. f) Sagital cut.

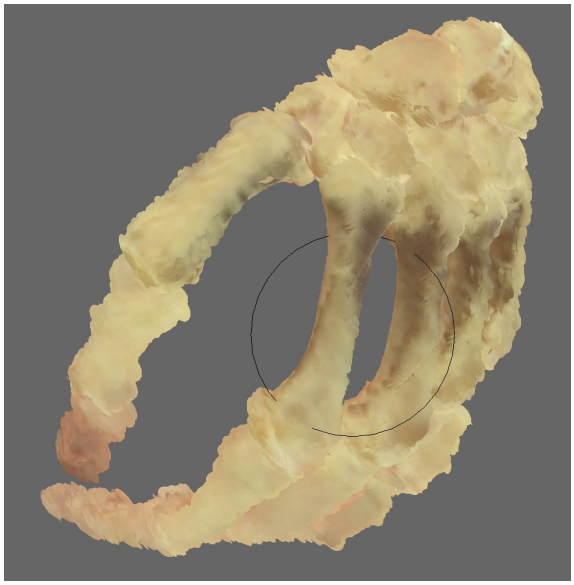


Figure 5.18: Precision Inspection Sphere. Inside the sphere surfels are represented at level n whereas outside it they are shown at level $n-2$.

ments a Precision Inspecting Sphere, consisting of a sphere which acts as a "resolution magnifying lens". The surfels inside the sphere are rendered at the finest level, while the rest of them are rendered at a coarse level. In Figure 5.18 all the bones of the hand are displayed at level $n-2$ except for part of two metacarpals which are displayed at level n. Video 5 in `http://www.cs.upc.edu/~jsurinach/Thesis/Videos.html` shows an interactive moving Precision Inspecting Sphere.

We can appreciate the similarities between the reconstructed organ and a real one as it is shown in Figure 5.19. In the original dataset the organ is still inside the body. Therefore, the shape of the real, surgically extracted organ can show some differences to the reconstructed one.

5.4.5 Memory requirements

In Table 5.6 the compression ratios of some representative organs from Table 5.4 are described. The first column shows the number of voxels in the model, the second one is the size in bytes of the Surfel Octree and the last column is the bandwidth usage measured in bits/voxel. The octree size column includes

Figure 5.19: Reconstructed and real pancreas.

Organ	**voxels**	**octree size**	**bandwidth**
Calcaneus	2311218	412844	1.429
Abductor Digiti Minimi	4789325	322433	0.539
Flexor Digitorum Tendon	30684420	203895	0.053
Metatarsal 1st	1600452	197398	0.987
Cuboid	318934	116740	2.928
Interosseous Plantar 3rd	308826	66680	1.727
Phalanx Cartil Mid 4th	192	288	12.005

Table 5.6: Compression ratios for several organs of the foot. Columns are: number of voxels in the model, Surfel Octree size in bytes, compression ratio in bits/voxel.

Color	Surfels		Octree	Textures	
method	n	n-2	structure	PNG 256	JPEG 128
Gradient	18118	1023	618	-	-
PTM	8236	465		58667	3182

Table 5.7: Storage requirements for the Forest Octree of the foot model of two surfel color methods. All values are expressed in KiloBytes.

the octree structure (the bit stream) and levels n and $n-2$ of surfels, as they are the most requested levels in our examples. The image of the organ is not included since it is sent only once per session.

The compression ratio depends on the shape of organs. Notice that compression values are usually below 2 bits per voxel. The value of 12.005 belongs to an small organ with few voxels.

Memory occupation of the Forest Octree of the foot model is shown in Table 5.7. Two surfel color methods are analyzed: Gradient of Intensity method in the first row and Projective Texture Mapping in the second row. The first two columns show the storage requirements of surfels in levels n and $n-2$. Next column is the size of the octree structure. Obviously, this value is the same for both methods. Two last columns express the size of textures used in the PTM method. Two options are included: png compression with 256×256 pixel images and jpeg compression at 128×128. This table illustrates the relative proportions between the different data structures that are transmitted over the network. Observe that the total memory requirements for the PTM option with small jpeg images and surfels at levels n and $n-2$ is lower than 13 MBytes for a total of 96 organs.

The construction of the Surfel Octree is done in preprocessing time. The complete foot took 265.35 seconds to complete. The rendering of the client achieves 60 fps when all organs are displayed at level $n-2$ and 13 fps when all are at level n. In situations where some organs are shown at level n and the rest at $n-2$ (as when the Precision Inspecting Sphere is in use) the frame rate is about 30, depending on the balance of number of surfels rendered at each level.

5.5 Discussion of the proposed architecture

The proposed architecture has been implemented on a client-server system. The Server is a portable computer with an Intel Core i7 CPU with 4 cores at

2 GHz and 4 GBytes of RAM, running the Windows 7 S.O. The client is a Nexus 10 tablet with a resolution of 2560 × 1600 pixels, 16 MBytes of internal storage, a dual-core A15 CPU and a quad-core Mali T604 GPU, running the Android 5.1.1.

The transmission between the Client and the Server uses a 24 MBps WiFi connection. This is the best configuration to check communication efficiency in our system. Servers placed remotely on the Internet produce variable -and worse- time results, depending on the network speed in those precise moments.

Several practical test cases have been made to analyze the performance of the overall architecture. The tests consists in the transmission of the scenes shown in figure 5.20, with several levels of complexity. Scenes consists in: a) phalanges of the 2nd finger of the right hand, b) sectioned kidney, c) pancreas, d) first four cervical vertebrae and d) section of the foot. As a curiosity, note the gap of the annular pancreas, a genetic anomaly that consists in the complete surrounding of the duodenum by the pancreatic tissue.

Table 5.8 contains the number of organs and surfels in the scenes of Figure 5.20. Surfels are detailed for levels n, $n-2$ and for the whole octree. The amount of data sent and transference times of the five scenes when levels n and $n-2$ are transmitted, are registered in Tables 5.9 and 5.10.

Columns in Table 5.9 describe the following elements: *admin* characterizes administrative data sent to the client, as the bounding box and the list of organs; *oct.n* and *oct.n-2* consist in the octree structure plus the surfels of the given level; *textures* is the texture occupation when compressed using the JPEG format; the last two columns contain the total storage occupation when the given level is represented.

Columns in Table 5.10 detail the transmission time of the previous elements. In addition, bandwidth is the ratio between the total storage values of Table 5.9 and the total time.

Given that administrative data contains a list of organs, its size depends only on the number of organs of the scene, not on the size of the organs. The same occurs to textures: since the dimensions of every image is the same for all organs, texture occupancy is determined also by the number of organs. Images of the foot are 3.3 times bigger than those of the vertebrae, although foot contains less surfels than vertebrae. The memory occupied by surfels is obviously connected linearly with the size of the organs, the number of organs and the level transmitted.

Time of transmission is affected by unpredictable circumstances that affect the network. Results show the average of ten trials. This fact explains that similar storage values -e.g. administrative data of kidney and pancreas- obtain different transmission times. But in a closed network this is a relative small variation when bigger values are involved and overall results keep their validity.

scene	organs	surfels		
		level n	level n-2	all levels
Finger	3	25502	1450	33558
Kidney	1	131219	7171	171483
Pancreas	1	192887	10375	251264
Foot	30	225311	12418	295120
Vertebrae	6	230874	13330	303978

Table 5.8: Number of organs and surfels in the test scenes from Figure 5.20.

scene	storage				total	
	admin	oct.n	oct.n-2	textures	n	n-2
Finger	248	108197	6151	292486	400931	298885
Kidney	156	555710	30356	108653	664519	139165
Pancreas	156	816422	43921	125612	942190	169689
Foot	1534	958582	52678	2577830	3537946	2632042
Vertebrae	259	979154	56596	782269	1761682	839124

Table 5.9: Storage of the test scenes from Figure 5.20. Storage is expressed in bytes.

The transmission of the administrative data and textures, which occurs once at the beginning of the session, represents a small fraction of communication time; the biggest part corresponds to the octree -with the exception of textures in scenes with many organs-. Tests display a delay of less than thirteen seconds in the worst case. This transmission is produced at the beginning of the session and every time the user modifies the level of the octrees, which occurs sporadically. On the other hand, interaction is smooth and achieves a reasonable performance, see `http://www.cs.upc.edu/~jsurinach/Thesis/Videos.html`, video 3.

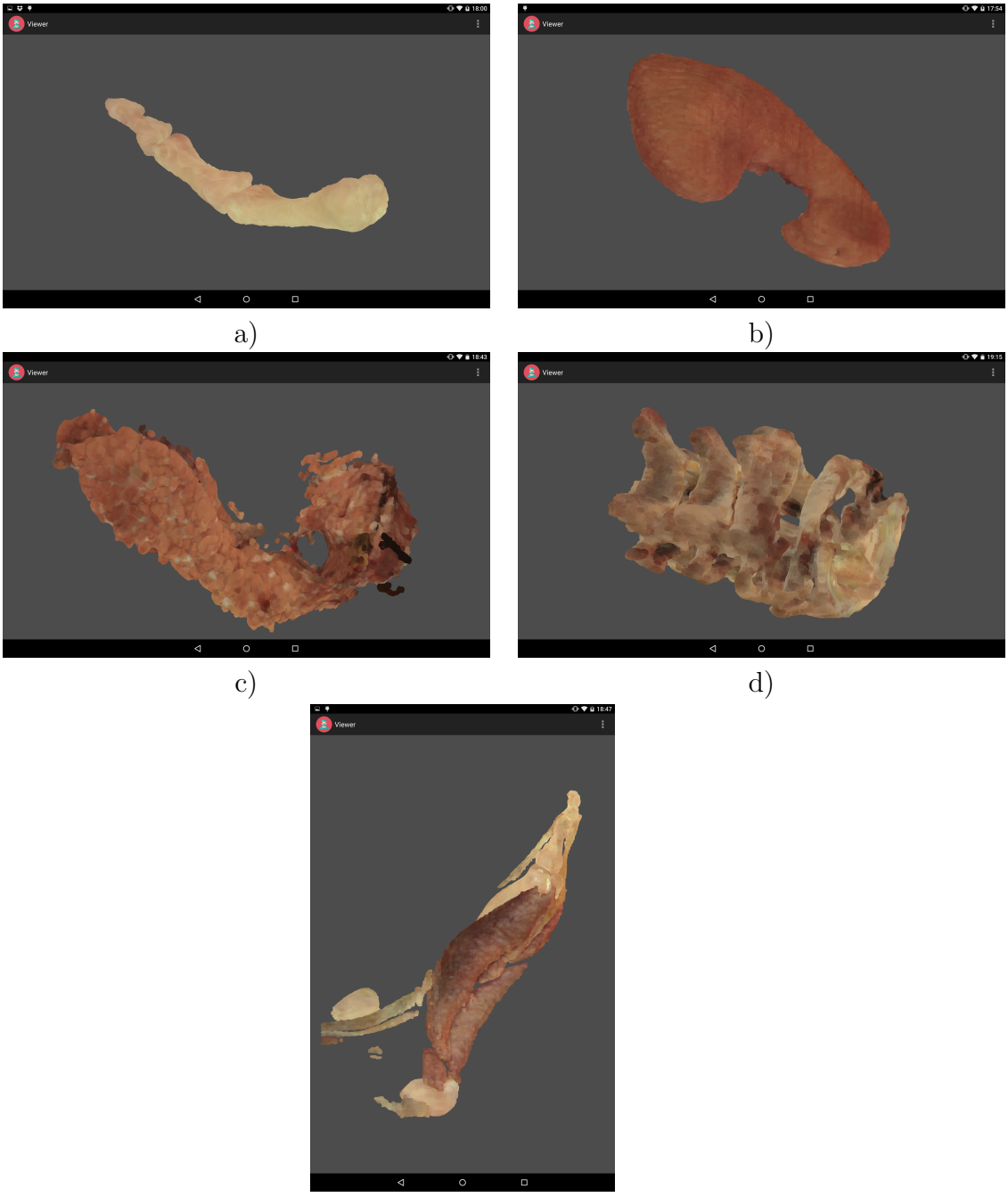


Figure 5.20: Images used for tablet testing. a) Phalanges of the 2nd finger of the right hand. b) Sectioned kidney. c) Pancreas. d) First four cervical vertebrae. e) Sectioned foot.

scene	transmission time				total		bandwidth	
	admin	oct.n	oct.n-2	textures	n	n-2	n	n-2
Finger	160	1294	74	3059	4513	3293	710.76	726.17
Kidney	115	5769	315	1128	7012	1558	758.16	714.61
Pancreas	128	8515	458	1310	9953	1896	757.30	715.93
Foot	368	12271	674	33000	45639	34042	620.16	618.53
Vertebrae	180	11273	652	9006	20459	9838	688.85	682.32

Table 5.10: Transmission time and bandwidth of the test scenes from Figure 5.20. Time is expressed in milliseconds and bandwidth in bits/second.

5.6 Conclusions

While Chapters 3 and 4 are devoted to the Forest Octree construction which takes place during the preprocessing phase, in this Chapter the complete framework of the Atlas has been presented, making explicit the roles of the Client and the Server parts and the interaction between them. Several examples and their performance are also studied.

The proposed scheme is client-driven. The Server contains the Forest Octree at the beginning of a session with the Client device. The user sends successive requests to the Server with a list of Organs along with the desired precision level. The transferred data is kept at a minimum since the octree structures and textures are sent only once. The interaction is designed to be performed at an interactive rate. Browsing of the model is performed exclusively in the Client.

Volume inspection is available at any moment through textured planar sections, as described in Section 5.1. Rotation and zooming operations with these planar sections can also be performed autonomously in the client, as discussed in Section 5.4.

Three interactive tools are presented. The section plane performs a cut in the rendered model and fills in the gap with an image obtained from the raw model, thus permitting the inspection of the inner structures of the organs. The Precision Inspecting Sphere allows a close inspection of a particular region of the inspected model by rendering with the most detailed surfels the geometry that lies inside a sphere. And finally, the selective rendering of organs chosen by the user in the scene.

A number of tests and examples carried out during the writing of this thesis are presented. The results are analyzed, emphasizing the visual appearance and memory consumption and pointing out the strengths and also the weaknesses

of our proposal.

Chapter 6

Conclusions and Future Research

This chapter presents the main conclusions of the thesis in Section 6.1. A number of avenues for future research are then listed in Section 6.2.

6.1 Conclusions

The main objective of this thesis has been to enrich the user experience on interactive visualization of Anatomical Atlases in low-cost portable devices by studying new representation schemes adapted to transmission, reconstruction and visualization in those devices.

The contributions of this work are summarized below:

Constrained Surfels

Surfels are well-suited data structures for representing locally organ boundaries modeled with segmented volumetric samples. We have proposed a new type of Surfels that has a compact representation and is faithful to the colors of the model.

Its compactness is based on an efficient encoding of its supporting plane. The plane is defined with a tetrahedron-based parameterization (Connected-Cubes Plane Parameterization, *CCPP*) that uses a quantized encoding of the values of a linear scalar field in four reference points. With this parameterization, the amount of data to encode a plane is reduced to 27 bits.

A preprocessing algorithm has been proposed to fair organ boundaries by smoothing the inaccuracies of the model due to binary segmentation. The algorithm approximates a smooth representation of the organ boundaries by the generation of a set of high-resolution surfels.

To maintain color fidelity, an analysis and comparison of four shading models of surfels has been made: a monochromatic shading of the surfel, a gradient variation of the intensity of its color, a projection of six points of the surfel to the underneath image and a projective texture mapping. The last method is the most convenient for its realistic color approximation and also for its low memory occupation.

Surfel Octrees

In Chapter 4 a hierarchical structure is proposed for the convenient storage of organ boundaries. This structure, the Surfel Octree, represents a single organ. The whole model with multiple organs is represented by a forest of octrees. The octree is populated by surfels, one in each node. The surfels in leaves are the high-resolution surfels. An algorithm is proposed to produce the surfels in inner nodes by a simplification process based on the surfels of children nodes.

Surfels get advantage of the octree structure since part of their geometry can be inferred from a recursive traversal of the octree. Therefore, this part is not required to be coded and the storage requirements of the surfel are reduced.

Every level of the octree constitutes a level of detail of the organ. Consequently, the octree becomes a multiresolution model which can be accessed per level or with a dynamic front, allowing multiple rendering possibilities.

In this scheme, the hierarchy structure is encoded as a bit stream, and the surfel's associated data can be encoded as a compact array of 4-Byte elements.

This representation enables progressive transmission of surfels, where only selected levels of surfels are transmitted.

Interactive inspection in client-server systems

We propose a scheme specially designed to work in a client-server environment, with a fat client approach. The medical model is preprocessed and a forest of Surfel Octrees is created. The Server stores the forest and sends it to a client in a low-cost portable device, which is responsible for rendering the surfels.

Medical experts are used to examining the inner structure of organs with planar cuts. A tool has been designed to perform planar cuts in the visualized organs. The client defines the plane which is generated in the server and sent back to the client at interactive speeds.

Using the multiresolution scheme, only surfels required by the client are transmitted from the server. Therefore, the client's memory only has to store the visible surfels. This progressive approach reduces both the net traffic and the time interval from a data request to its arrival.

The basic user interaction -translating, rotating, zooming and panning of the model- is performed entirely in the client without the intervention of the server. The change of the visualization resolution level of an organ produces a request to the server.

Given the previous conclusions, we can assert that the main hypothesis stated in Section 1.2 that a discrete and hierarchical surfel representation is efficient and suitable for the interaction with anatomic atlases in distributed applications with low-end client devices has been fully achieved.

6.2 Future Research

There are some aspects of the presented work which can be further investigated and extended:

The icosahedron-like distribution of point of views to produce the textures of an organ, guarantees a distribution of textures that uniformly covers all the organ boundary only if the organ has a spherical shape. In the case of not-so-circular forms, e.g. a vein, an elongated muscle or a tendon, some textures may include many more surfels than another, resulting in a reduced size of the surfels in the texture and worse accuracy of texture coordinates. A previous analysis of the shape of the organ can lead to other distributions, resulting in a more uniform distribution of voxels per texture.

The proposed projective texture mapping approach is still rather verbose, when compared to the storage requirements of the octree and surfels. We plan to investigate algorithms based on adaptive texture synthesis and bidirectional texture functions with the objective of reducing memory needs while adapting to organ color appearances.

Neighbor surfels are likely to use the same texture, which is also used by their parent surfels. This results in very similar images, even in different LODs. Taking advantage of this property, the use of a multiresolution scheme where the level displayed depends on the local curvature may be studied: it uses coarser surfels in mainly flat areas and high-resolution surfels in regions of high curvature radius. Unnecessary surfels will be sent to the renderer in places where geometric detail is not required.

The render quality of surfels can be improved by using elliptical weighted average (EWA) surface splatting, although our sampled data is distributed in a regular grid.

A study to collect the opinion of potential users on the application should be carried on to make a proper and useful design of the interface with the Client.

Two interaction tools have been presented, but the habitual use of the Atlas can lead the experts to add new requirements to their interactive sessions. Further investigation of interaction tools in accordance to medical expert needs remains an open subject.

Bibliography

[1] *Acland's Video Atlas of Human Anatomy* https://aclandanatomy.com Ed. Lippincott Williams & Wilkins, 2013.

[2] A.M.R. Agur, A.F. Dalley, *Grant's Atlas of Anatomy*, 13th edition. Edited by A.M. Gilroy, B.R. MacPherson and L.M. Ross. ISBN 978-1-60406-952-5. Ed. Thieme, 2009.

[3] *Anatomia Creations* http://anatomia-creations.com

[4] *The BioDigital Human* https://www.biodigital.com, 2014.

[5] Blausen Medical / Medical Animations. http://blausen.com

[6] C.D. Clemente, *A Regional Atlas of the Human Body*, 6th edition, 2011. ISBN 978-1-58255-889-9

[7] G. Fabricio, *Tabulae Pictae*, plate 112.10. Biblioteca Nazionale Marciana, San Marco, Venezia.

[8] H. Gray, H.V. Carter, *Anatomy Descriptive and Surgical*. Publisher: John W. Parker and Son, London, 1859.

[9] K.H. Höhne, S. Gehrmann, T. Nazar, A. Petersik, B. Pflesser, A. Pommert, U. Schumacher, U. Tiede, *VOXEL-MAN 3D Navigator: Upper Limb (Arm and Hand) Regional and Radiological Anatomy*. ISBN 978-3-540-21010-8. Springer Electronic Media, New York, 2008.

[10] K.H. Höhne, A. Petersik, B. Pflesser, A. Pommert, K. Priesmeyer, M. Riemer, T. Schiemann, R. Schubert, U. Tiede, M. Urban, H. Frederking, M. Lowndes, J. Morris, *VOXEL-MAN 3D Navigator: Brain and Skull. Regional, Functional, and Radiological Anatomy*. ISBN 978-3-642-01211-2. Springer Electronic Media, New York, 2009.

[11] Interactive Institute Swedish ICT, *Inside Explorer*. https://www.tii.se/projects/insideexplorer, 2012.

[12] J.A. Juanes, A. Prats, M.L. Lagandara, J.M. Riesco, *Application of the "Visible Human Project" in the field of anatomy: A review*. European Journal of Anatomy. 7(3):147-159, 2003.

[13] *Kenhub*, https://www.kenhub.com, 2015.

[14] C-Y. Lin, D.T. Chen, R.B. Loftin, J. Chen, E.L. Leiss, *Interacting with Visible Human Data Using an ImmersaDesk*. Proceedings of the IEEE Virtual Reality 2002 (VR'02), pp. 267-268, 2002. DOI: 10.1109/VR.2002.996531

[15] *K.P. Moses, J.C. Banks, P.B. Nava, D.K. Petersen, Atlas of Clinical Gross Anatomy*, 2nd edition. ISBN 978-0-323-07779-8. Ed. Elsevier Saunders, 2013.

[16] V. Monclús, J. Díaz, I. Navazo, P.P. Vàzquez, *The Virtual Magic Lantern: An Interaction Metaphor for Enhanced Medical Data Inspection*. Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology, pp. 119-122, 2009. DOI: 10.1145/1643928.1643955

[17] F.H. Netter, *Atlas of Human Anatomy*, 6th edition. Ed. Elsevier, 2005. ISBN 978-1455704187

[18] Nebraska Medicine. http://www.nebraskamed.com/health-library/3d-medical-atlas

[19] R. Perez, J. Rojo-Manaute, J. Vaquero, J.M. Martinez, F. Chana, *3D surgical printing in preoperative planning and pre contoured plates for acetabular fractures. Do it yourself*. Journal of the International Foundation for Computer Assisted Radiology and Surgery, ISCAS. 10(1):84-85, 2015.

[20] F. Paulsen, J. Waschke, *Sobotta Atlas of Human Anatomy. General Anatomy and Musculoskeletal System*, 15h edition. Ed. Elsevier, 2011.

[21] J.W. Rohen, E. Ltjen-Drecoll, C. Yokochi, *Color Atlas of Anatomy: A Photographic Study of the Human body*. ISBN 9781582558561. 7th edition. Ed. Lippincott Williams & Walkins, 2011.

[22] V. Spitzer, M.J. Ackerman, A.L. Scherzinger, D. Whitlock, *The Visible Human Male: A Technical Report*. Journal of the American Medical Informatics Association, Vol 3(2), 1996.

[23] K. Sakata, J. Kondo, T. Nishimura, Y. Maeda, K. Morita, *Three-dimensional elastic liver model and hepatic surgery and interventional approach using simulation images*. Journal of the International Foundation for Computer Assisted Radiology and Surgery, ISCAS. 10(1):80-81, 2015.

[24] M. Schuenke, E. Schulte, U. Schumacher, *Atlas of Anatomy*, 2nd. edition. Edited by A.M. Gilroy, B.R. MacPherson, L.M. Ross. Ed. Thieme, 2009. ISBN 978-1-60406-952-5

[25] *Touch of Life Technologies (ToLTech)* http://www.toltech.net http://anatomia-creations.com/

[26] *Visible Body*, http://www.visiblebody.com. Argosy Publishing, Inc.

[27] *Zygote Body* http://www.zygote.com Zygote Media Group, 2014.

[28] C. Andújar, P. Brunet, J. Esteve, E. Monclús, I. Navazo, A. Vinacua, *Robust face recovery for hybrid surface visualization*. 8th International Workshop in Vision, Modeling and Visualization, VMV'03, Munich, Germany 2003.

[29] A. Doi, A. Koide, *An efficient method of triangulating equivalued surfaces by using tetrahedral cells*. IEICE Transactions Communication E74, 1, 214224, 1991.

[30] Interactivity is the key, *W. Hibbard, D. Santek*. Proceedings of the ACM Workshop on Volume visualization (VVS'89), Chapel Hill, pp. 39-43, 1989. DOI: 10.1145/329129.329356

[31] W.E. Lorensen, H.E. Cline, *Marching Cubes: A High Resolution 3d Surface Construction Algorithm*. Proceedings of SIGGRAPH, pp. 163-169, 1987.

[32] Efficient Ray Tracing of Volume Data, *Marc Levoy*. ACM Transactions on Graphics, 9(3):245-261, 1990 DOI:10.1145/78964.78965

[33] Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation, *P. Lacroute, M. Levoy*. Proceedings of the SIGGRAPH '94, pp. 451-458, 1994. DOI: 10.1145/192161.192283

[34] SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm, *L.A. Westover*. Ph.D. thesis, technical report TR91-029, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1991.

[35] Automatic restoration of polygon models, *S. Bischoff, D. Pavic, L. Kobbelt*. ACM Transactions on Graphics (TOG), 24(4):1332-1352, 2005. DOI: 10.1145/1095878.1095883

[36] J. Esteve, P. Brunet, A. Vinacua, *Approximation of a variable density cloud of points by shrinking a discrete membrane*. Computer Graphics Forum, Volume 24, Number 4 ,pp 791-808, 2005.

[37] H.S. Kang, B.H. Kim, J.W. Ryu, S.H. Hong, H.W. Chung, S.Y. Cho, Y.H. Kim, S.I. Hwang, D.K. Jeong, Y.G. Shin, *The Visible Man: 3D Interactive Musculoskeletal Anatomic Atlas of the Lower Extremity*. Radiographics, 20(1):279-286, 2000. DOI: 10.1148/radiographics.20.1.g00ja23279

[38] U. Tiede, T. Shiemann, K. Hoehne, *High quality rendering of attributed volume data*. Proceedings of the IEEE Visualization Conference, pp. 255-262, 1998.

[39] Y.J. Zhang, C. Bajaj, T.J.R. Hughes, *Automatic 3D Mesh Generation Method for Domains with Multiple Materials*. International Meshing Roundtable IMR 2007, pp. 367-386, 2007.

[40] Y.J. Zhang, C. Bajaj, B.S. Sohn, *3D finite element meshing from imaging data*. Computer methods in applied mechanics and engineering 194(48):5083-5106, 2005.

[41] O. Soldea, A. Ekin, D.F. Soldea, D. Unay, M. Çetin, A. Erçil, M.G. Uzunbaş, Z. Firat, M. Cihangiroglu, *Segmentation of Anatomical Structures in Brain MR Images Using Atlases in FSL - A Quantitative Approach*. Proceedings of the IEEE 20th International Conference on Pattern Recognition (ICPR), pp. 2592-2595, 2010. DOI: 10.1109/ICPR.2010.635

[42] F.L. Seixas, A. Silveira de Souza, *Anatomical Brain MRI Segmentation Methods:Volumetric Assessment of the Hippocampus*. 17th International Conference on Systems, Signals and Image Processing (IWSSIP), 17:247-251, 2010.

[43] C. Andújar, P. Brunet, A. Chica, J. Rossignac, I. Navazo, A. Vinacua, *Optimizing the topological and combinational complexity of isosurfaces*. Computer-Aided Design, 37 (8), pp. 847-857, 2005.

[44] F. Allamandri, P. Cignoni, C. Montani, R. Scopigno, *Adaptlively Adjusting Marching Cubes Output to Fit a Trilinear Reconstruction Filter*, Proceedings of the Eurographics Workshop in Blaubeuren, Germany April 20-22 Part I, pp. 25-34, 1998. DOI: 10.1007/978-3-7091-7517-0_3

[45] E.V. Chernyaev, *Marching cubes 33: construction of topologically correct isosurfaces*. Technical Report CERM CM 95-17, CERN, 1995.

[46] C.A. Dietrich, C. Scheidegger, J. Schreiner, J.L.D. Comba, L.P. Nedel, C.T. Silva, *Edge Transformations for Improving Mesh Quality of Marching Cubes*. IEEE Transactions on Visualization and Computer Graphics, 15(1):150-159, 2008. DOI: 10.1109/TVCG.2008.60

[47] H. Freeman, *Computer processing of line-drawing images*. ACM Computing Surveys, 6:57-97, 1974.

[48] S.F.F. Gibson, *Using Distance Maps for Accurate Surface Representation in Sampled Volumes*. Volume Visualization Symposium IEEE 1998, pp. 263-270, 1998.

[49] T. Ju, F. Losasso, S. Schaefer, J. Warren, *Dual Contouring of Hermite Data*. Proceedings of ACM SIGGRAPH, pp 339-346, 2002.

[50] P. Kobbelt, M. Botsch, U. Schwanecke,H.P. Seidel, H.-P., *Feature-Sensitive Surface Extraction from Volume Data*. Proceedings of ACM SIGGRAPH, pp. 57-66, 2001.

[51] C:K. Manikandtan, S. Resmi, S. Sibi, R.R. Kumar, G.S. Harikumaran Nair *Templated marching cubes A low computation approach to surface rendering*, IEEE Recent Advances in Intelligent Computational Systems (RAICS), pp. 268-271, 2013. DOI: 10.1109/RAICS.2013.6745485

[52] C: Montani, R. Scateni, R.Scopigno, *Discretized Marching Cubes*. Proceedings of the Conference on Visualization. pp. 281-287. IEEE Computer Society Press, 1994.

[53] G.M. Nielson, B. Hamann, *The asymptotic decider: resolving the ambiguity in marching cubes*. Proceedings of the 2nd conference on Visualization '91. pp. 83-91, 1991.

[54] G.M. Nielson, *Dual Marching Cubes*. Proceedings of the conference on Visualization 2004. pp. 489-496, 2004.

[55] T.S. Newman, H. Yi, *A survey of the marching cubes algorithm*. Computers & Graphics, 30(5):854-879, 2006. DOI:10.1016/j.cag.2006.07.021

[56] X. Wang, Y. Niu, L.Tan, S-X. Zhang *Improved marching cubes using novel adjacent lookup table and random sampling for medical object-specific 3D visualization*. Journal of Software, 9(10):2528-2537, 2014.

[57] C. Andújar, P. Brunet, M. Fairén, I. Navazo, A. Vinacua, *A Topological Comparison of Surface Extraction Algorithms*. Internal report from MOVING research group (UPC) LSI-05-66-R, 2005.

[58] L. Buatois, G. Caumon, B.Levy, *GPU Accelerated Isosurface Extraction on Tetrahedral Grids*. Proceedings of the International Symposium on Visual Computing (ISVC), 4291:383-392, 2006.

[59] L.S. Chen, G.T. Herman, R.A. Reynolds, J.K. Udupa, *Surface shading in the cuberille environment*. Computer Graphics and Applications 10, pp 33-43, 1985.

[60] G.M. Nielson, *Dual Marching Tetrahedra: Contouring in the Tetrahedronal Environment*. Advances in Visual Computing, Springer Berlin/Heidelberg. pp. 183-194, 2004.

[61] V. Pascucci, *Isosurface Computation Made Simple: Hardware Acceleration, Adaptive Refinement and Tetrahedral Stripping*. IEEE TVCG Symposium on Visualization '04. pp. 293-300, 2004.

[62] M. Scholz, J. Bender, C. Dachsbacher, *Real-Time Isosurface Extraction With View-Dependent Level of Detail and Applications*. Computer Graphics Forum, 24(1):103-115, 2015. DOI: 10.1111/cgf.12462

[63] K. Weiss, L.D. Floriani, *Simplex and Diamond Hierarchies*. Computer Graphics Forum, 30(8):2127-2155, 2011.

[64] S. Bischoff, L.P. Kobbelt, *Isosurface Reconstruction with Topology Control*. Pacific Graphics 2002 Proceedings, pp. 246-255, 2002.

[65] S. Bischoff, L.P. Kobbelt, *Extracting consistent and manifold interfaces from multi-valued volume data sets*. Bildverarbeitung fr die Medizin, 2006.

[66] A. Chica. *Visibility-Based Feature Extraction From Discrete Models*. ACM Symposium on Solid and Physical Modeling, pp. 347-352, 2008.

[67] J. Chen, X. Jin, Z. Deng, *GPU-based polygonization and optimization for implicit surfaces*. The Visual Computer: International Journal of Computer Graphics archive. 31(2):119-130, 2015. DOI: 10.1007/s00371-014-0924-7

[68] A. Chica, J. Williams, C. Andujar, P. Brunet, I. Navazo, J. Rossignac, A. Vinacua, *Pressing: Smooth Isosurfaces with Flats from Binary Grids* Computer Graphics Forum, 27(1):36-46, 2006.

[69] M. Desbrun, M. Meyer, P. Schrder, A.H. Barr, *Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow* Proceedings of SIGGRAPH 1999, pp. 317324, 1999.

[70] T.R. Jones, F. Durand, M. Desbrun, *Non-iterative, feature-preserving mesh smoothing*. Proceedings of SIGGRAPH, 22(3):943-949, 2003. DOI: 10.1145/1201775.882367

[71] v. Lempitsky *Surface Extraction from Binary Volumes with Higher-Order Smoothness* IEEE Conference on Computer Vision and Pattern Recognition, pp. 1197-1204, 2010.

[72] Y. Ohtake, A.G. Belyaev, *Mesh optimization for polygonized isosurfaces*. Compututer Graphics Forum, 20(3):368376, 2001. DOI: 10.1111/1467-8659.00529

[73] X. Sun, P.L. Rosin, R.R. Martin, F.C. Langbein, *Fast and Effective Feature-Preserving Mesh Denoising*. IEEE Transactions on Visualization and Computer Graphics, 13(5):925-938, 2007. DOI: 10.1109/TVCG.2007.1065

[74] G. Taubin, *A Signal Processing Approach to Fair Surface Design*. Proceedings of SIGGRAPH 1995, pp. 351-358, 1995. DOI: 10.1145/218380.218473

[75] T. Tasdizen, R. Whitaker, P. Burchard, S. Osher, *Geometric Surface Smoothing via Anisotropic Diffusion of Normals*. VIS'02, Proceedings of the conference on Visualization, 2002.

[76] J. Wang, Z. Yu, *A Novel Method for Surface Mesh Smoothing: Applications in Biomedical Modeling* Proceedings of the 18th International Meshing Roundtable. pp. 195-210, 2009.

[77] Z. Zhong, X. Guo, W. Wang, B. Levy, F. Sun, Y. Liu, W. Mao, *Particle-Based Anisotropic Surface Meshing* ACM Transaction on

Graphics, Proceedings of SIGGRAPH 103 Conference, 32(4):99, 2013. DOI: 10.1145/2461912.2461946

[78] P. Agarwal, L. Guibas, A. Nguyen, D. Russel, L. Zhang, *Collision detection for deforming necklaces*, Computational Geometry, 28(2-3 special issue):137-163, 2004. DOI: 10.1016/j.comgeo.2004.03.008

[79] J. Arvo, D. Kirk, *A Survey of Ray Tracing Acceleration Techniques*. An Introduction to Ray Tracing, Glassner. Academic Press Ltd. London, UK, pp. 201-262, 1989. ISBN:0-12-286160-4

[80] J. Arvo, *Linear Time Voxel Walking for Octrees*. Ray Tracing News, Feb 1988.

[81] P. Berman, B. DasGupta, S. Muthukrishnan, Exact Size of Binary Space Partitionings and Improved Rectangle Tiling Algorithms. SIAM J. Discrete Math. 15(2):252-267, 2002. DOI: 10.1137/S0895480101384347

[82] J.L. Bentley, *Multidimensional Binary Search Trees Use for Associative Searching*. Communications of the ACM, 18(9):509-517, 1975. DOI: 10.1145/361002.361007

[83] G. Bernstein, D. Fussell, *Fast, Exact, Linear Booleans*. Proceedings of the Symposium on Geometry Processing (SGP'09), pp. 1269-1278, 2009.

[84] A. Bardera, M. Feixas, I. Boada, J.Rigau, M. Sbert, *Medical Image Registration Based on BSP and Quad-tree Partitioning*. Proceedings of the Third International Workshop on Biomedical Image Registration 2006 (LNCS 4057), 2006. DOI: 10.1007/11784012_1

[85] N. Chin, S. Feiner, *Near Real-Time Shadow Generation Using BSP trees*. Proceedings of SIGGRAPH 89, 23(3):99-106, 1989.

[86] R.A. Finkel, J.L. Bentley, *Quad trees: A Data Structure for Retrieval on Composite Keys*. Acta Informatica, 4(1):19, 1974.

[87] H. Fuchs, Z.M. Kedem, B Naylor *Predetermining Visibility Priority in 3-D Scenes*. Proceedings of SIGGRAPH 79, pp 175-181, 1979.

[88] H. Fuchs, Z.M. Kedem, B. Naylor, *On Visible Surface Generation by A Priori Tree Structures* Proceedings of the 7th annual conference on Computer graphics and interactive techniques, SIGGRAPH '80, pp 124-133, 1980. DOI: 10.1145/965105.807481

[89] H.W. Jensen, J.C. Niels, *Photon maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects*. Computers and Graphics 19 (2), pp. 215-224, 1995.

[90] M.C. Gill, A.Y. Zomaya, *A Cell Decomposition Based Collision Avoidance Algorithm for Robot Manipulators*. Journal of Cybernetics and Systems , 29(2):113-135, 1998. DOI: 10.1080/019697298125759

[91] G.M. Morton, *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. IBM Ltd. Technical Paper, Morton1966, 1966.

[92] B.F. Naylor, *Interactive Solid Modeling Using Partitioning Trees*. Proceedings on Graphics Interface, pp. 11-18, 1992.

[93] J. Romero, director, *Quake*. GT Interactive Bethesda Softworks, 1996.

[94] S. Rusinkiewicz, M. Levoy, *QSplat: a multiresolution point rendering system for large meshes*. Proceedings of the 27th annual conference on Computer graphics and interactive techniques, SIGGRAPH '00, pp. 343-352, 2000. DOI:10.1145/344779.344940

[95] S.M. Rubin, T. Whitted, *A 3-Dimensional Representation for Fast Rendering of Complex Scenes*. Proceedings of the 7th annual conference on Computer graphics and interactive techniques, SIGGRAPH '80, pp. 110-116, 1980. DOI: 10.1145/965105.807479

[96] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Ed. Morgan-Kaufmann, San Francisco, CA, 2006. ISBN-13: 978-0123694461

[97] R.A. Schumacker, R. Brand, M. Gilliland, W. Sharp, *Study for Applying Computer Generated Images to Visual Simulation*. AFHRL-TR-69-14, U.S. Air Force Human Resources Laboratory, 1969.

[98] I. Wald, V. Havran, *On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N)*. IEEE Symposium on Interactive Ray Tracing, pp. 61-69, 2006. DOI: 10.1109/RT.2006.280216

[99] K. Zhou, Q. Hou, R. Wang, B. Guo, *Real-time KD-tree Construction on Graphics Hardware* Proceedings of ACM SIGGRAPH Asia 2008, 27(5), Article No. 126, 2008. DOI: 10.1145/1409060.1409079

[100] J. A. Bærentzen, *Octree-based Volume Sculpting*. IEEE Visualization, 1998.

[101] J. Baert, A. Lagae, P. Dutré, *Out-of-Core Construction of Sparse Voxel Octrees*. Computer Graphics Forum, 33:220227, 2014. DOI: 10.1111/cgf.12345

[102] I. Boada, I. Navazo, R. Scopigno, *Multiresolution volume visualization with a texture-based octree*. The Visual Computer 17(3):185-197, 2001.

[103] B. Cabral, N. Cam, J. Foran, *Accelerated volume rendering and tomographic reconstruction using texture mapping hardware*. Proceedings of the Symposium on Volume Visualization, VVS 94, pp. 9198, 1994.

[104] F. Ferstl, R. Westermann, C. Dick, *Large-Scale Liquid Simulation on Adaptive Hexahedral Grids*. IEEE Transactions on Visualization and Computer Graphics, 20(10):1405-1417, 2014

[105] A. S. Glassner, *Space Subdivision For Fast Ray Tracing*. IEEE Computer Graphics and Applications, 4(10):1522, 1984. DOI: 10.1109/MCG.1984.6429331

[106] A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard, *OctoMap:An Efficient Probabilistic 3D Mapping Framework Based on Octrees*. Autonomous Robots, 34(3):189-206, 2013. DOI: 10.1007/s10514-012-9321-0

[107] J. Jessup, S. Givigi, A. Beaulieu, *Robust and efficient multi-robot 3D mapping with octree based occupancy grids*. IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 3996-4001, 2014. DOI: 10.1109/SMC.2014.6974556

[108] T. Ju, *Robust repair of polygonal models*. Proceedings of the ACM SIGGRAPH '04, pp. 888-895, 2004. DOI: 10.1145/1186562.1015815

[109] A. Knoll, I. Wald, S. Parker, C. Hansen, *Interactive Isosurface Ray Tracing of Large Octree Volumes*. IEEE Symposium on Interactive Ray Tracing, pp. 115-124, 2006. DOI: 10.1109/RT.2006.280222

[110] F. Losasso, F. Gibou, R. Fedkiw, *Simulating water and smoke with an oc.tree data structure*. ACM SIGGRAPH '04 Papers, pp. 457-462, 2004. DOI: 10.1145/1186562.1015745

[111] Y. Livnat, C.D. Hansen, *View Dependent Isosurface Extraction*. Proceedings of IEEE Visualization 98, pages 175180, 1998. DOI: 10.1109/VISUAL.1998.745300

[112] S. Lefebvre, S. Hornus, F. Neyret, *Octree Textures on the GPU*, chapter in GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation. NVidia, 2004. ISBN-13: 978-0321335593

[113] S. Laine, T. Karras, *Efficient sparse voxel octrees*. Proceedings of th Symposium International on 3D Graphics Games, pp. 5563, 2010.

[114] D.J.R. Meagher, *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. Rensselaer Polytechnic Institute. Image Processing Laboratory. IPL-TR-80-111, 1980.

[115] B. Mora, J.P. Jessel, R. Caubet, *Visualization of Isosurfaces with Parametric Cubes*. Computer Graphics Forum, 20(3):377-384, 2001. DOI: 10.1111/1467-8659.00530

[116] M. Pharr, R. Fernando, *Gpu gems 2: Programming techniques for high performance graphics and general-purpose computation*. Addison-Wesley Professional, 2005. ISBN:0321335597

[117] H. Samet, *Implementing ray tracing with octrees and neighbor finding*. Computers and Graphics, 13(4):44560, 1989.

[118] K. Sung, *A DDA octree traversal algorithm for ray tracing*. Proceedings of the Eurographics 91, pp. 7385, 1991.

[119] J.Wilhelms, A. Van Gelder, *Octrees for faster isosurface generation*. ACM Transactions on Graphics, 11(3):201227, 1992. DOI: 10.1145/130881.130882

[120] O. Wilson, A. VanGelder, J. Wilhelms, *Direct Volume Rendering via 3D Textures*. Technical report, University of California Santa Cruz, CA, USA, 1994.

[121] C. Andújar, P. Brunet, D. Ayala, *Topology-Reducing Surfaces Simplification Using a Discrete Solid Representation*. ACM Transactions on Graphics, 21(3):1-18, 2002.

[122] C. Andújar, M. Fairen, P. Brunet, V. Cebollada, *Error-bounded simplification of topologically-complex assemblies*. 4th Workshop on Multiresoluion and Geometric Modelling, MINGLE'03, Cambridge 2003.

[123] M. Botsch, L., Kobbelt, *Multiresolution Surface Representation Based on Displacement Volumes*. Computer Graphics Forum 22(3) (Proceedings of Eurographics 2003), 483-491, 2003.

[124] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, R. Scopigno, *Adaptive TetraPuzzles. Efficient Out-of-core Construction and Visualization of Gigantic Polygonal Models*. ACM Transactions on Graphics, Proceedings of SIGGRAPH 2004. 23(3):796-803, 2004.

[125] C. Crassin, F. Neyret, S. Lefebvre, E. Eisemann, *Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering*. Proceedings of the Symposium on Interactive 3D Graphics and Games (I3D 09), ACM, pp. 1522, 2009.

[126] T.A. Funkhouser, C.H. Séquin, *Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments*. Computer Graphics (SIGGRAPH '93), pp. 247-254, 2003.

[127] T. Fogal, A. Schiewe, J. Krüger, *An analysis of scalable GPU-based ray-guided volume rendering*. IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV), pp. 4351, 2013.

[128] E. Gobetti, M. Marton, *Layered Point Clouds. A Simple and Efficient Multiresolution Structure for Distributing and Rendering Gigantic Point-Sampled Models*. Computers & Graphics, 28(6):815-826, 2004.

[129] E. Gobetti, F. Marton, *Far Voxels. A Multiresolution Framework for Interactive Visualization of Huge Complex 3D Datasets on Commodity Graphics Platforms*. ACM Transactions on Graphics, Proceedings of SIGGRAPH 2005.24(3):878-885, 2005.

[130] E. Gobbetti, F. Marton, J. Iglesias Guitián, *A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets*. Proceedings of The Visual Computer 24(7-9):797806, 2008.

[131] S. Guthe, W. Strasser, Advanced techniques for high-quality multi-resolution volume rendering. Computers & Graphics 28(1):5158, 2004

[132] H. Hoppe, *Progressive Meshes*. Proceedings of ACM SIGGRAPH'96. pp 99-108, 1996.

[133] L. Hu, P.V. Sander, H.Hoppe, *Parallel View-Dependent Refinement of Progressive Meshes*. Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 169-176, 2009. DOI: 10.1145/1507149.1507177

[134] M. Kraus, K. Bürger, *Interpolating and downsampling RGBA volume data*. Proceedings of Vision, Modeling, and Visualization, pp. 323332, 2008.

[135] E. Lamar, B. Hamann, K.I. Joy, *Multiresolution techniques for interactive texture-based volume visualization*. Proceedings of the Conference on Visualization (VIS 99), IEEE Computer Society Press, pp. 355361, 1999.

[136] P. Lindstrom, D. Koller,W. Ribarsky, L.F. Hodges, N. Faust,G.A. Turner, *Real-Time, Continuous Level of Detail Rendering of Height Fields*. ACM Transactions on Graphics, Proceedings of SIGGRAPH 96, pp. 109-118, 1996.

[137] M. Marinov, M. Botsch, L.P. Kobbelt, *GPU-Based Multiresolution Deformation Using Approximate Normal Field Reconstruction*. ACM Journal of Graphics Tools 12(1), pp. 27-46, 2007.

[138] M.A. Otaduy, M.C. Lin, *CLODs: Dual Hierarchies for Multiresolution Collision Detection*. Proceedings of the Eurographics Symposium on Geometry Processing, pp. 94-101, 2003.

[139] M. Pauly, L.P. Kobbelt, M. Gross, *Point-Based Multiscale Surface Representation*. ACM Transactions on Graphics, 25(2):177-193, 2006.

[140] R. Sicat, M. Hadwiger, J. Krüger, *Sparse PDF volumes for consistent multi-resolution volume rendering*. IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Visualization) 20(12):24172426, 2014.

[141] C. Wang, J. Gao, L. Li, H.W. Shen, *A multiresolution volume rendering framework for large-scale time-varying data visualization*. Proceedings of the Fourth Eurographics/IEEE VGTC conference on Volume Graphics, Eurographics Association, pp. 1119, 2005.

[142] M. Weiler, R. Wertermannm C. Hansen, K. Zimmermann, T. Ertl, *Level-of-detail volume rendering via 3D textures*. Proceedings of the IEEE Symposium on Volume Visualization (VVS 00), ACM, pp. 713, 2000.

[143] Y.S. Wang, C. Wang, T.Y. Lee, K.L. Ma, *Feature-preserving volume data reduction and focus+context visualization*. IEEE Transactions on Visualization and Computer Graphics 17(21):171181, 2011.

[144] X. Xu, E. Sakhaee, A. Entezari, *Volumetric data reduction in a compressed sensing framework*. Computer Graphics Forum 33(3):111120, 2014.

[145] M. Agus, E. Gobbetti, J.A. Iglesias, F. Marton, *Split-Voxel: A Simple Discontinuity-Preserving Voxel Representation for Volume Rendering*. IEEE/EG International Symposium on Volume Graphics, 2010.

[146] M. Balsa, E. Gobbetti, J.A. Iglesias, M. Makhinya, F. Marton, R. Pajarola, S.K. Suter, *A Survey of Compressed GPU-Based Direct Volume Rendering State of the Art Report*. Eurographics STARS, 2013.

[147] L. Campoalegre, I. Navazo, P. Brunet., *Gradient Octrees: A New Scheme for Remote Interactive Exploration of Volume Models*, Computer-Aided Design and Computer Graphics (CAD/Graphics), International Conference on, pp. 306-313, 2013.

[148] N. Fout, M. Kwan-Liu, *Transform Coding for Hardware-accelerated Volume Rendering*. IEEE Transactions on Visualization and Computer Graphics, 13(6):1600-1607, 2007.

[149] S. Guthe, W. Strasser, *Real-time decompression and visualization of animated volume data*. Proceedings of the conference on Visualization 2001, pp. 349-356, 2001.

[150] I. Ihm, S. Park, *Wavelet-based 3D compression scheme for interactive visualization of very large volume data*. Computer Graphics Forum, 18:3-15, 1999.

[151] S. Muraki, *Approximation and rendering of volume data using wavelet transforms*. Proceedings of the 3rd Conference on Visualization, pp. 21-28, 1992.

[152] P. Ning, L. Hesselink, *Vector quantization for volume rendering*. Proceedings of the workshop on Volume Visualization'92, pp. 69-74, 1992.

[153] S.K. Suter, M. Makhynia, R. Pajarola, *TAMRESH: Tensor Approximation Multiresolution Hierarchy for Interactive Volume Visualization*. Eurograpics Conference on Visualization (Euro Vis), 32(3):151-160, 2013.

[154] J. Schneider, R. Westermann, *Compression domain volume rendering*. Proceedings of the 14th IEEE Visualization 2003, pp. 293-300, 2003. DOI: 10.1109/VISUAL.2003.1250385

[155] M. Alexa, J. Behr, D. Cohen-or, S. Fleishman, D. Levin, C.T. Silva, *Point Set Surfaces*. IEEE Visualization 2001, pp. 2128, 2001. ISBN 0-7803-7200-x

[156] B. Adams, P. Dutré, *Interactive boolean operations on surfel-bounded solids*. Proceedings of ACM SIGGRAPH 2003, ACM Transactions on Graphics, 22(3):651-656, 2003. DOI: 10.1145/882262.882320

[157] E.E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.D. thesis, University of Utah, Salt Lake City, 1974.

[158] R.L. Carceroni, K.N. Kutulakos, *Multi-View Scene Capture by Surfel Sampling: From Video Streams to Non-Rigid 3D Motion, Shape & Reflectance*. International Journal of Computer Vision. 49(2-3):175-214, 2002. DOI: 10.1023/A:1020145606604

[159] G. Guennebaud, M. Gross, *Algebraic point set surfaces*. Proceedings of the ACM SIGGRAPH '07. Article No. 23, 2007

[160] N. Greene, P.S. Heckbert, *Creating raster Omnimax images from multiple perspective views using the elliptical weighted average filter*. IEEE Computer Graphics and Applications, 6(6):21-27, 1986. DOI: 10.1109/MCG.1986.276738

[161] G. Guennebaud, M. Paulin, *Efficient screen space approach for Hardware Accelerated Surfel Rendering*. IEEE Signal Processing Society, Vision, Modeling and Visualization. pp 485-495, 2003.

[162] P. Heckbert, *Fundamentals of Texture Mapping and Image Warping*. Master thesis. University of California at Berkeley, Department of Electrical Engineering and Computer Science, June 17 1989.

[163] P. Mavridis ,G. Papaioannou, *High Quality Elliptical Texture Filtering on GPU*. Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D), pp. 23-30, 2011. DOI: 10.1145/1944745.1944749

[164] H. Pfister, M. Zwicker, J. van Baar, M. Gross, *Surfels: Surface Elements as Rendering Primitives*. Proceedings of SIGGRAPH 2000, pp. 335-342, 2000.

[165] J. Stückler, S. Behnke, *Multi-Resolution Surfel Maps for Efficient Dense 3D Modeling and Tracking*. Journal of Visual Communication and Image Representation 25(1):137-147, 2014.

[166] M. Slomp, I. Daribo, R. Furukawa, R. Sagawa, S. Hiura, N. Asada, H. Kawasaki, *Hardware-Accelerated Geometry Instancing for Surfel and Voxel Rendering of Scanned 4D Media*. 11th International Conference on Quality Control by Artificial Vision (QCAV), 9 pages, 2013.

[167] M. Zwicker, H. Pfister, J. van Baar, M. Gross, *Surface Splatting*. SIGGRAPH 2001, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 371378, 2001.

[168] S.P. Callahan, L. Bavoil, V. Pascucci, C.T. Silva, *Progressive Volume Rendering of Large Unstructured Grids*, IEEE Transactions on Visualization and Computer Graphics, pp. 1307-1314, 2006.

[169] E. Gobbetti, J.A. Iglesias, F. Marton, *COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks*. EuroVis 2012, Computer Graphics Forum, 31(3):1315-1324, 2012.

[170] E.J. Luke, C.D. Hansen, *Semotus Visum: A Flexible Remote Visualization Framework*. IEEE Visualization, VIS 2002, pp. 61-68, 2002. DOI: 10.1109/VISUAL.2002.1183758

[171] H. Zhou, H. Qu, Y. Wu, M-Y. Chan, *Volume Visualization on Mobile Devices*. Pacific Conference on Computer Graphics and Applications 2006, pp. 76-84, 2006.

[172] L. Campoalegre, P. Brunet, I. Navazo, *Interactive Visualization of Medical Volume Models in Mobile Devices*. Personal and Ubiquitous Computing, Springer-Verlag, 2012.

[173] T. Capin, K. Pulli, T. Akenine-Moller, *The state of the art in mobile graphics research*. IEEE Computer Graphics and Applications, 28(4):7484, 2008. DOI: 10.1109/MCG.2008.83

[174] J. Díaz, V. Monclús, I. Navazo, P. Vàzquez, *Adaptive Cross-Sections of Anatomical Models*. Computer Graphics Forum (Proc. Pacific Graphics), Forum 31(7):2155-2164, 2012.

[175] T. Fogal, J. Krüger, *Tuvok, an Architecture for Large Scale Volume Rendering*. Proceedings of the 15th International Workshop on Vision, Modeling, and Visualization (VMV'10), 2010.

[176] F. Lamberti, C. Zunino, A. Sanna, A. Fiume, M. Maniezzo, *An Accelerated Remote Graphics Architecture for PDAS*. Proceedings of the 8th International Conference on 3D Web Technology. ACM, pp. 55-61, 2003.

[177] M. Mobania, Lin Feng, *Mobile Visualization of Biomedical Volume Datasets*. Journal of Internet Technology and Secured Transactions (JITST), 1(1), 2012.

[178] M. Moser, D. Weiskopf, *Interactive Volume Rendering on Mobile Devices*. Workshop on Vision, Modelling, and Visualization (VMV'08), 2008.

[179] J.M. Noguera, J.R. Jiménez, C.J Ogáyar, R.J. Segura, *Volume Rendering Strategies on Mobile Devices*. International Conference on Computer Graphics Theory and Applications (GRAPP), 2012.

[180] P.P. Vàzquez, M. Balsa, *Practical volume rendering in mobile devices*. International Simposium on Visual Computing, Lecture Notes in Computer Science, 7431:708-718, 2012.

[181] S.C. Daly, *Visible differences predictor: An Algorithm for the Assessment of Image Fidelity*. SPIE Proceedings, Human Vision, Visual Processing, and Digital Display III, 1666:179-205, 1992. DOI:10.1117/12.135952

[182] R. Mantiuk, K. Myszkowski, H-P Seidel, *Visible Difference Predicator for High Dynamic Range Images*. Proceedings of IEEE International Conference on Systems, Man and Cybernetics, pp. 2763-2769, 2004

[183] R. Mantiuk, K.J. Kim, A.G. Rempel, W. Heidrich, *HDR-VDP-2: A Calibrated Visual Metric for Visibility and Quality Predictions in All Luminance Conditions*. ACM Transactions on Graphics, Proceedings of SIGGRAPH'11, 30(4):40:1-40:14, 2011. DOI: 10.1145/2010324.1964935

[184] C. Andújar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, A. Vinacua, *Computing Maximal Tiles and Application to Impostor-Based Simplification*. Computer Graphics Forum, 23(3):401-410, 2004.

[185] D. Antonovic, *Review of the Hough Transform Method, With an Implementation of the Fast Hough Variant for Line Detection*. Technical Report TR663. The Trustees of Indiana University, 2008.

[186] L. ap Cenydd, N.W John, M. Bloj, A. Walter, N.I. Phillips, *Visualizing the Surface of a Living Human Brain*. IEEE Computer Graphics and Applications (Special Issue - Modeling and Rendering Material Appearance), 32(2):55-65, 2012.

[187] , R.O. Duda, P.E. Hart, *Use of the Hough Transform to detect lines and curves in pictures*. Communications of the ACM, 15(1):11-15, 1972. DOI: 10.1145/361237.361242

[188] X. Décoret, F. Durand, F.X. Sillion, J Dorsey, *Billboard Clouds for Extreme Model Simplification*. ACM Transactions on Graphics, 22(3):689-696, 2003.

[189] P.V.C. Hough, *Method and Means for Recognizing Complex Patterns*. U.S. Patent No. 3069654, 1962 DOI: 10.1111/j.1467-8659.2007.01039.x

[190] Q. Meyer, J. Süßmuth, G. Sußner, M. Stamminger, G. Greiner, *On Floating-Point Normal Vectors*. Proceedings of the 21st Eurographics Symposium on Rendering (EGSR'10), 29(4):1405-1409, 2010 DOI: 10.1111/j.1467-8659.2010.01737.x

[191] L.S. Liebovitch, T. Tibor, *A fast algorithm to determine fractal dimensions by box counting*. Physics Letters A, 141(8-9):386-390, 1989. DOI: 10.1016/0375-9601(89)90854-2

[192] G. Nielson, G. Graf, R. Holmes, A. Huang, M. Phielipp, *Shrouds: Optimal separating surfaces for enumerated volumes*. EG-IEEE TCVG Symposium on Visualization, pp. 7584, 2003.

[193] J. Surinyac, P. Brunet, *A Client-Server Architecture for the Interactive Inspection of Segmented Volume Models*. XXIII Congreso Español de Informática Gráfica (CEIG), pp. 99-108, 2013.

[194] Jordi Surinyac, Pere Brunet, *Surfel Octrees: A New Scheme for Interactive Inspection of Anatomy Atlas in Client-Server Applications*. XXV Congreso Español de Informática Gráfica (CEIG), pp. 61-70, 2015. DOI: 10.2312/ceig.20151201